Signalogic DSP Software

Users Guide

Signalogic, Inc.
Dallas, Texas

Rev. 2
June, 1997

The software described in this document is furnished under license agreement. The software may be used or copied only under the terms of this license agreement (see section 3.1.1).

Copyright © 1994-1997 Signalogic, Inc. All rights reserved.

No part of this manual may be photocopied or reproduced in any form without prior written consent from Signalogic, Inc.

U.S. Government: If Licensee is acquiring the software on behalf of any unit or agency of the United States Government, the following shall apply:

(a) for units of the Department of Defense:
RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data Clause at DFARS 252.227-7013.

(b) for any other unit or agency:
NOTICE -- Notwithstanding any other lease or license agreement that may pertain to, or accompany the delivery of, the computer software and accompanying documentation, the rights of the Government regarding use, reproduction and disclosure are as set forth in Clause 52.227-19(c)(2) of the FAR.

Contractor/manufacturer is Signalogic, Inc., 9617 Wendell @ Skillman, Dallas, TX 75243.

Hyperception is a registered trademark of Hyperception, Inc.
Signalogic and DSPower are registered trademarks, and Real-Time Convolver is a trademark of Signalogic, Inc.
MATLAB is a registered trademark of The MathWorks, Inc.
National Instruments is a registered trademark, and NIDSP is a trademark of National Instruments, Inc.
Borland is a registered trademark of Borland International, Inc.
Microsoft is a registered trademark, and MS-DOS and Windows are trademarks of Microsoft Corporation.
Analog Devices, AT&T, Motorola, NEC, and Texas Instruments are registered trademarks of their respective holders. ADSP-21xx, WE DSP32x, DSP5600x, DSP9600x, 77220, and TMS320xx are trademarks of their respective holders.
Crystal Semiconductor is a registered trademark of Crystal Semiconductor, Inc.
PC is a trademark of IBM Corporation.
All other trademarks belong to their respective holders.

Signalogic, Inc.
9617 Wendell @ Skillman
Dallas, TX 75243
214-343-0069

Signalogic DSP Software Users Guide, June 1997
Design and Implementation: Jeff Brower.

Programming: Jeff Brower , Varsha Shamabhat.
Documentation: Jeff Brower, Varsha Shamabhat, Hanh Nguyen.

Contents

1 System Information 1

Overview 1
1.1 Summary of Software Quick Reference 2
1.2 System Requirements Quick Reference 2
1.3 System Considerations Quick Reference 3
1.4 Documentation Quick Reference 3
1.5 Installation Quick Reference 4
1.6 Troubleshooting Quick Reference 5

2 Technical Reference 7

Overview 7
2.1 Block Quick Reference 8
2.2 Function Quick Reference 8
2.3 DSP/Analog Hardware Quick Reference 9
2.4 Waveform File Format Quick Reference 9
2.5 Definitions Quick Reference 9
2.6 Message Quick Reference 10
2.7 Error and Status Condition Quick Reference 10

3 Using Hypersignal®-Macro Series Software 11

Overview 11
3.1 Command-Line Interface to Hypersignal 14
3.1.1 Command-Line Options for Hypersignal 16
3.1.2 Command-Line Examples Of Running Hypersignal 21
3.2 Keyboard and Mouse Interface In Hypersignal User Mode 22
3.3 Online Help In Hypersignal User Mode 26
3.4 Main Menu in Hypersignal User Mode 27
3.4.1 Menu Selection in Hypersignal User Mode 30
3.4.2 Quit/Exit in Hypersignal User Mode 31

3.4.3 User Setup Area in Hypersignal 32
3.4.4 System Status Area in Hypersignal 35
3.5 Data-Entry Editor in Hypersignal User Mode 43
3.5.1 Waveform Summary Display in Hypersignal 47
3.5.2 Popup Directory Display in Hypersignal 49
3.6 Printing Menus and Waveform Displays in Hypersignal User Mode 53
3.7 Running External Programs in Hypersignal 54
3.7.1 DOS Shell in Hypersignal User Mode 56
3.8 Configuration Files in Hypersignal 58
3.9 Creating Waveform Files in Hypersignal 62
3.9.1 Creating Complex Waveform Files in Hypersignal 63
3.9.2 Creating Multichannel Waveform Files in Hypersignal 64
3.9.3 Creating Frequency Domain Waveform Files 65
3.10 Finding Waveforms in Hypersignal User Mode 66
3.11 Running Macro Programs in Hypersignal 71
3.11.1 Running a Macro Program From the Hypersignal Macro Manager Menu 72
3.11.2 Running a Macro Program From Outside of Hypersignal 75
3.11.3 Running a Hypersignal Macro Program from Another Macro 76
3.12 Getting Waveforms In and Out of Hypersignal User Mode 77
3.13 Displaying Time Domain Waveforms in Hypersignal User Mode 77
3.14 Interactive Waveform Displays in Hypersignal User Mode 78
3.15 FFT Generation in Hypersignal User Mode 78
3.16 Displaying Frequency Domain Waveforms in Hypersignal User Mode 78
3.17 Waveform Generation and Mathematics in Hypersignal User Mode 78
3.18 DSP/Acquisition Hardware in Hypersignal User Mode 79
3.19 Filter Design in Hypersignal User Mode 79
3.20 Applying Filters in Hypersignal User Mode 79
3.21 Creating Macro Programs and Functions in Hypersignal User Mode 79

4 Using DSPower®-HWLib and DSPower-SPLib Software 81

Overview 81
4.1 Architecture of DSPower-HWLib and DSPower-SPLib Software 82
4.1.1 Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software 84
4.1.2 Callback Messages in DSPower-HWLib and DSPower-SPLib Software 85
4.1.3 Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software 87
4.1.4 Performance Considerations in DSPower-HWLib and DSPower-SPLib Software 91
4.2 Creating Programs for DSPower-HWLib and DSPower-SPLib 95
4.2.1 Hardware Library and Signal Processing Library Access 97
4.2.2 Include Files for DSPower-HWLib and DSPower-SPLib 98
4.2.3 .LIB Files for DSPower-HWLib and DSPower-SPLib 99 100
4.2.4 Example Code: DSPower-HWLib and DSPower-SPLib Software 101
4.3 DSPower-HWLib Function Overview 113
4.3.1 Hardware and Processor Control 114
4.3.2 DSP Program File Downloading 119
4.3.3 Hardware Interrogation 120
4.3.4 DSP Memory Data Transfers 127
4.3.5 Executing High-Level Functions 130
4.4 DSPower-SPLib Function Overview 131
4.5 Acquiring Data Using DSPower-HWLib Software 131

5 Using DSP Source Code Interface Software 137

Overview 137
5.0.1 DSP Source Code Interface Documentation 139
5.0.2 DSP Source Code Interfaces for Hypersignal 140
5.0.3 DSP Source Code Interfaces for Hypersignal in Engine Mode 147
5.1 Texas Instruments TMS320C25 Source Code Interface 148
5.1.1 Texas Instruments TMS320C25 Source Code Interface for Hypersignal 149

5.1.2 TMS320C25 Source Code Interface File List for Hypersignal 152
5.1.3 TMS320C25 Source Code Modification Process for Hypersignal 155
5.2 AT&T WE DSP32 Source Code Interface 158
5.2.1 AT&T WE DSP32 Source Code Interface for Hypersignal 159
5.2.2 DSP32 Source Code Interface File List for Hypersignal 160
5.2.3 DSP32 Source Code Modification Process for Hypersignal 160
5.3 Motorola DSP5600x Source Code Interface 164
5.3.1 Motorola DSP5600x Source Code Interface for Hypersignal 164
5.3.2 DSP5600x Source Code Interface File List for Hypersignal 168
5.3.3 DSP5600x Source Code Modification Process for Hypersignal 171
5.4 AT&T WE DSP32C Source Code Interface 174
5.4.1 AT&T WE DSP32C Source Code Interface for Hypersignal 175
5.4.2 DSP32C Source Code Interface File List for Hypersignal 178
5.4.3 DSP32C Source Code Modification Process 180
5.5 Texas Instruments TMS320C3x Source Code Interface 185
5.5.1 Texas Instruments TMS320C3x Source Code Interface for Hypersignal 186
5.5.2 TMS320C3x Source Code Interface File List for Hypersignal 190
5.5.3 TMS320C3x Source Code Modification Process for Hypersignal 192
5.6 NEC 77220 Source Code Interface 198
5.6.1 NEC 77220 Source Code Interface for Hypersignal 198
5.6.2 77220 Source Code Interface File List for Hypersignal 200
5.6.3 77220 Source Code Modification Process for Hypersignal 202
5.7 Analog Devices ADSP-21xx Source Code Interface 206
5.7.1 Analog Devices ADSP-21xx Source Code Interface for Hypersignal 207

5.7.2 ADSP-21xx Source Code Interface File List for Hypersignal 211
5.7.3 ADSP-21xx Source Code Modification Process for Hypersignal 213
5.8 Motorola DSP96002 Source Code Interface 217
5.8.1 Motorola DSP96002 Source Code Interface for Hypersignal 217
5.8.2 DSP96002 Source Code Interface File List for Hypersignal 221
5.8.3 DSP96002 Source Code Modification Process for Hypersignal 224
5.9 AT&T WE DSP3210 Source Code Interface 228
5.9.1 AT&T WE DSP3210 Source Code Interface for Hypersignal 228
5.9.2 WE DSP3210 Source Code Interface File List for Hypersignal 232
5.9.3 WE DSP3210 Source Code Modification Process for Hypersignal 234
5.10 Texas Instruments TMS320C4x Source Code Interface 238
5.10.1 Texas Instruments TMS320C4x Source Code Interface for Hypersignal 239
5.10.2 TMS320C4x Source Code Interface File List for Hypersignal 242
5.10.3 TMS320C4x Source Code Modification Process for Hypersignal 245
5.11 Texas Instruments TMS320C5x Source Code Interface 248
5.11.1 Texas Instruments TMS320C5x Source Code Interface for Hypersignal 249
5.11.2 TMS320C5x Source Code Interface File List for Hypersignal 252
5.11.3 TMS320C5x Source Code Modification Process for Hypersignal 255

6 Using DSPower®-Block Diagram Software 259

Overview 259
6.1 Getting Started 261
6.1.1 Installation and Checkout 263
6.1.2 Online Help in DSPower 263
6.1.3 Block Selector 264
6.1.4 Worksheet 268
6.1.5. Waveform Locator 269

6.1.6 Style/Setup Locator 271
6.2 Options in DSPower 273
6.2.1 Program Options 273
6.2.2 Worksheet Options 273
6.2.3 Diagram Options 275
6.2.4 Display Options 278
6.2.5 Help Options 279
6.3 Configuring DSPower 280
6.3.1 Directory Configuration 281
6.3.2 DSP/Analog Hardware Configuration 284
6.3.3 DSP Engine Configuration 284
6.3.4 Online Editor Configuration 287
6.3.5 Online Debugger Configuration 288
6.3.6 Block Libraries Configuration 289
6.3.7 Operator Mode (Stand-Alone Apps) Configuration 290
6.4 Creating Block Diagrams 291
6.4.1 Diagrams 291
6.4.2 Connecting Blocks 291
6.4.3 Creating Projects 291
6.4.4 Multiple Diagrams 292
6.4.5 Saving and Retrieving Block Diagrams 292
6.4.6 Executing Block Diagrams 292
6.5 Adding User Defined Blocks 292
6.5.1 Creating and Adding Blocks 292
6.5.2 Creating derivative Blocks 292
6.5.3 Adding Code Blocks 293
6.5.4 Modifying Controls 293
6.6 Interactive Operation 293
6.6.1 Run One Block 293
6.6.2 Loading Waveforms 293
6.6.3 Loading Setup and Style Files 293
6.6.4 Viewing and Editing Template and Generated Source Files 293
6.7 Tools in DSPower 294
6.7.1 Operator Mode 294
6.7.2 Make Stand-Alone Icon 294
6.7.3 Block Wizard 294
6.7.4 Waveform Headitor 294
6.8 Debugging Diagrams 294

7 Using Engine Manager 291

Overview 291

7.1 Engine Manager Access 292
7.2 Controlling DSP Engines 293
7.2.1 Opening DSP Engines 294
7.2.2 Sending DSP Engine Commands 296
7.3 Executing Functions in DSP Engines 297
7.4 Registering Callback Messages from DSP Engines 299
7.5 Error Handling in Engine Manager 315
7.5.1 DLL Error Handling in Engine Manager 316
7.5.2 Engine Error Handling 317
7.5.3 Error/Status Window in Engine Manager 320

8 Using Hardware Manager 325

Overview 325
8.1 Hardware Manager Access 326
8.2 Hardware Selector 327
8.2.1 Selecting DSP/Analog Hardware Types 329
8.2.2 DSP/Analog Hardware Configuration 330
8.2.3 Hardware Values and Variables 340
8.3 Hardware Setup File 341
8.3.1 Hardware Setup File Fields 344
8.3.2 Hardware Setup File Access 355
8.4 Error Handling in Hardware Manager 359
8.4.1 DLL Error Handling in Hardware Manager 360

Glossary of Terms 323

Index 327

Glossary of Terms

.ide

Filename extension used for Borland C/C++ v4.5 project files.

.mak

Filename extension used for Microsoft Visual C/C++ v1.5 project files.

.prj

Filename extension used for Borland C/C++ v3.1 project files.

API

Application Programming Interface. Refers to interface presented by DLL or other Windows object to external applications.

DDE

Dynamic Data Exchange. A Windows 3.x compatible method of transferring data between applications which don't "know" about each other. Likely to be replaced by the faster and more reliable OLE method under Windows '95.

DLL

Dynamic Link Library. A library of routines which can be linked to any number of applications on demand, at run-time. The library is treated as an object, to which applications know how to interface, but not how it functions internally.

DRAM

Dynamic random access memory. Typically used as large amount of auxiliary memory; for a DSP/analog board, but sometimes used instead of SRAM if the DSP device has enough onchip SRAM. 4-5 times slower and less expensive than SRAM.

EX

Extended. Used to refer to extended functions or applications. Examples include the f_fftex and f_adex functions, and the Hypersignal-Macro EX software.

expanded memory

Additional, usually large, amount of memory added to a PC system, but accessed through a small (64 kbyte) segment in conventional memory space. No longer as prevalent with advent of systems configured with 4 Mbyte or more of standard (extended) memory.

extended memory

System memory located above 1024 kbyte. This is the memory referred to when a system is described as having 4 Mbyte or RAM, or 16 Mbyte of RAM, etc. installed.

IDE

Integrated Development Environment. A term commonly used by C/C++ compiler vendors such as Borland, Microsoft, and WATCOM to refer to their collective tools and utilities when presented under a unified graphical user-interface.

ISR

Interrupt Service Routine. Used frequently in discussing the structure and function of DSP source code.

MLS

Maximum Length Sequence. A pseudo-random noise sequence of digital 1s and 0s mathematically defined to not repeat for N-1 samples, where N= 2**^{M}, and M is the sequence order.

NDA

Non-Disclosure Agreement.

NRT

Non-Real-Time. Commonly used to refer to non-real-time versions of software. Hypersignal-Macro NRT is an example.

RT

Real-Time. Commonly used to refer to real-time versions of software.

OLE

Object Linking and Embedding. Refers to a Windows compatible method for applications which otherwise don't "know" each other to share information and data.

SRAM

Static random access memory. Typically used as main program and data memory for a DSP/analog board. 4-5 times faster and more expensive than DRAM.

VxD

Virtual Device Driver. A type of Windows device driver used to perform low-level and/or priveleged system functions or communication.

windows.h

Standard Microsoft Windows 3.1 include file needed by most C/C++ programs to provide common type definitions and constants..

Index

Error! No index entries found.

1 System Information

Overview

[@k information,, system]

This chaptersection is included here mainly as a quick reference to system information areas of other Signalogic DSP Software documentation. The intent behind presenting a quick reference here, in the Users Guide, is twofold:

1. provide a convenient reference to these very important topics from inside the Users Guide, especially for the online Help version of documentation, which will automatically jump to the appropriate help file after a quick reference is selected
2. not leave Users Guide readers wondering, even momentarily, as to the location of system information

Quick Reference Topics

Summary of Software Quick Reference

System Requirements Quick Reference

System Considerations Guide Quick Reference

Documentation Quick Reference

Installation Quick Reference

Troubleshooting Quick Reference

1.1 Summary of Software Quick Reference

[@k System Guide]

For an overall perspective of Signalogic DSP software, see Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components (in the System Guide online Help File).

For more information detailed information, but still in summary form, see the following sections in the Signalogic DSP Software System Guide:

1.2.1 Hypersignal®-Macro Series Software Summary

1.2.2 DSPower-HWLib Software Summary

1.2.3 DSP Source Code Interface Software Summary

1.2.4 DSPower-Block Diagram Software Summary

1.2.5 DSPower-C/C++ Code Generator Software Summary

1.2.6 Real-Time Convolver Software Summary

1.2 System Requirements Quick Reference

[@k System Guide]

The following topics are covered in Signalogic DSP Software System Guide section 1.3, System Requirements (in the System Guide online Help file):

Operating System Requirements

Processor Requirements

Memory (RAM) Requirements

Video / Graphics Requirements

Mouse Requirements

Math Coprocessor Requirements

Printer Requirements

DSP Engine Requirements

DSP/Analog Hardware Requirements

Windows C/C++ Development Tool Requirements

DSP Development Tool Requirements

1.3 System Considerations Quick Reference

[@k System Guide]

Closely related to the subject of System Requirements, but providing more discussion about performance, system compatibility, and system tradeoffs, is Signalogic DSP Software System Guide section 1.4, System Considerations

(in the System Guide online Help file), which covers the following:

1.4.1 Memory Considerations

1.4.2 Performance Considerations

1.4.3 Video and Display Considerations

1.4.4 DSP/Analog Hardware Considerations

1.4 Documentation Quick Reference

[@k System Guide]

See Signalogic DSP Software System Guide chapter 2, Documentation, (in the System Guide online Help file) for complete information, including the following sections:

2.1 Documentation Assumption of Previous Knowledge

2.2 Documentation Organization

2.3 Documentation Conventions

2.4 Documentation References and Additional Sources

1.5 Installation Quick Reference

[@k System Guide]

Signalogic DSP Software System Guide chapter 3, Installation, (in the System Guide online Help file) covers installation of all Signalogic DSP Software components, including the following topics:

3.1 Before You Install

3.1.1 License Agreement

3.1.2 Media and Disk Space Requirements

3.1.3 Media and Documentation You Should Have Received

3.2.2 Installing Hypersignal-Macro Series Software

3.2.2 Installing DSPower Series Software

3.2.3 Configuration and Checkout of DSPower-HWLib Software

3.2.4 Configuration and Checkout of DSPower-Block Diagram Software

3.2.5 Installing and Configuring Hypersignal Software For User Mode Operation

3.3 Release Notes

3.4 Directory Structure Chart

3.5 Uninstalling

1.6 Troubleshooting Quick Reference

[@k System Guide]

For detailed information in a "problem / suggestion" format, see the following topics sections in the Signalogic DSP Software System Guide online Help file:

4.1 Troubleshooting Installation Problems

4.2 Troubleshooting Video and Display Problems

4.3 Troubleshooting DSP/Analog Hardware Problems

2 Technical Reference

Overview

[@k reference,, technical]

This chaptersection is included here mainly as a quick reference to technical reference areas of other Signalogic DSP Software documentation. The intent behind presenting a quick reference here, in the Users Guide, is twofold:

1. provide a convenient reference to these very important topics from inside the Users Guide, especially for the online Help version of documentation, which will automatically jump to the appropriate help file after a quick reference is selected
2. not leave Users Guide readers wondering, even momentarily, as to the location of technical reference information

Quick Reference Topics

Block Quick Reference

Function Quick Reference

DSP/Analog Hardware Quick Reference

Waveform File Format Quick Reference

Definitions Quick Reference

DSP Engine Message Quick Reference

Error and Status Message Quick Reference

2.1 Block Quick Reference

[@k Reference Guide]

For detailed information in a "problem / suggestion" format, see the following Signalogic DSP Software System Guide sections (in the System Guide online Help File):

1.1 Code Blocks

1.2 Control Blocks

1.3 Display Blocks

1.4 Filter Blocks

1.5 Function Blocks

1.6 Instrument Blocks

1.7 Utility Blocks

2.2 Function Quick Reference

[@k Reference Guide]

For detailed information in a "problem / suggestion" format, see the following Signalogic DSP Software System Guide sections (in the System Guide online Help File):

2.1 DSP Engine Functions

2.2 DLL Functions

2.3 DSP Source Code Functions

2.3 DSP/Analog Hardware Quick Reference

[@k Reference Guide]

For detailed information in a "problem / suggestion" format, see the following Signalogic DSP Software System Guide sections (in the System Guide online Help File):

3.1 DSP/Analog Hardware Information

3.2 Board Designator, Driver ID, and Board Classification Table

3.3 Hypersignal DSP Source Code Variable List

2.4 Waveform File Format Quick Reference

[@k Reference Guide]

For detailed information in a "problem / suggestion" format, see the following Signalogic DSP Software System Guide sections (in the System Guide online Help File):

4.1 Hypersignal Waveform File Format

4.2 Microsoft .WAV Waveform File Format

4.3 MATLAB Waveform File Format

4.4 Acoustic/Audio Waveform File Formats

2.5 Definitions Quick Reference

[@k Reference Guide]

For detailed information in a "problem / suggestion" format, see the following Signalogic DSP Software System Guide sections (in the System Guide online Help File):

5.1 HWLISTENTRY

5.2 CONVERSIONINFO

2.6 Message Quick Reference

[@k Reference Guide]

For detailed information in a "problem / suggestion" format, see the following Signalogic DSP Software System Guide sections (in the System Guide online Help File):

6.1 DLL Messages

6.2 DSP Engine Messages

2.7 Error and Status Condition Quick Reference

[@k Reference Guide]

For detailed information in a "problem / suggestion" format, see the following Signalogic DSP

Software System Guide sections (in the System Guide online Help File):

7.1 DSPower-Block Diagram Error and Status Messages

7.2 DLL Error and Status Definitions

7.3 DSP Engine Error Messages

3 Using Hypersignal®-Macro Series Software

Overview

Some of the sections listed below are more or less reprinted from sections 3.1 through 3.10, and 4.1 through 4.10, of the Hypersignal-Macro Series Users Manual, but with the addition of information helpful in placing the Hypersignal-

Macro series software packages in perspective with other Signalogic DSP Software components. Examples include noting differences between User Mode and Engine Mode operation, and precise locations of subjects in all forms of Signalogic DSP Software documentation.

NOTE: Sections reprinted from Hypersignal-Macro Series Users Manual sections 3.1 through 3.10, and sections 4.1 through 4.10, may have material that copyrighted © 1985-1990 by Hyperception, Inc, and/or copyrighted © 1991-1995, by Signalogic, Inc.. All rights are reserved by their respective copyright holders.

Also, these sections provide an alternate source of online Help when using a Hypersignal-Macro series package under Windows, in addition to the native Hypersignal online Help facility. To conveniently access this help when operating Hypersignal software in user mode (see User Mode and Engine Mode, in Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components), do the following:

1. run the Hypersignal-Macro series software package (Hypersignal-Macro NRT, Hypersignal-Macro, Hypersignal-Macro EX, or Hypersignal-Acoustic) from its Windows icon in the Signalogic DSP Software group in the Program Manager window

2. once the Hypersignal software is running, use the Alt-Tab keyboard sequence to switch back to Progam Manager, leaving the Hypersignal software active

3. select this online Help section, and make it full screen

4. now use Alt-Tab to switch back to Hypersignal; use Alt-Tab again to re-activate the online Help when needed

The sections below assumes that you have already installed the Hypersignal software according to the instructions in Hypersignal-Macro Series Users Manual section 2.1, Installing Hypersignal, or in Signalogic DSP Software System Guide section 3.2.5, Installing and Configuring Hypersignal Software For User Mode Operation. These section can be divided into two basic groups:

- main Hypersignal user-interface
- accomplishing specific tasks in Hypersignal, using one or more of its functions, displays, and instruments

Assuming that a Hypersignal-Macro series software package is up and running, user-interface topics covered include executing Hypersignal with command-line parameters, initial setup and configuration once the software is running, and basic user-interface and system-interface options available in the software, such as keyboard and mouse interface, online Help, data-entry editor, printing screens and waveform displays, invoking external programs, invoking macro programs and functions, and locating waveform files.

Sections focusing on accomplishing specific tasks in Hypersignal have been organized into broad and general categories and thus cover most DSP analysis

achievable in Hypersignal. Because of this organization, some sections may cover more than one individual Hypersignal function or display. For a concise, item-by-item reference to all Hypersignal functions, displays, instruments, data-entry fields, and interactive display commands, see the "Hypersignal-Macro Series Software Reference Guide".

Main Hypersignal User-Interface

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

Accomplishing Specific Hypersignal Tasks

Getting Waveforms In and Out of Hypersignal User Mode

Displaying Time Domain Waveforms in Hypersignal User Mode

Interactive Waveform Displays in Hypersignal User Mode

FFT Generation in Hypersignal User Mode

Displaying Frequency Domain Waveforms in Hypersignal User Mode

Waveform Generation and Mathematics in Hypersignal User Mode

DSP/Acquisition Hardware in Hypersignal User Mode

Filter Design in Hypersignal User Mode

Applying Filters in Hypersignal User Mode

Creating Macro Programs and Functions in Hypersignal User Mode

3.1 Command-Line Interface to Hypersignal

To start Hypersignal software, first path to the hard disk drive and subdirectory containing Hypersignal executable files. Then, invoke one of the following batch files (which one you have depends on your software version) by typing:

Table 1 Command-Line Batch Files to Run Hypersignal Software

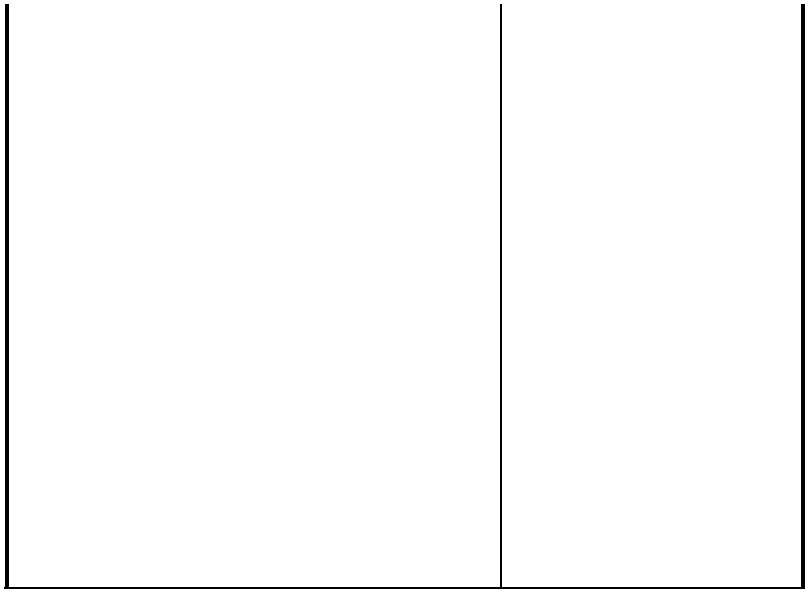
	Batch File	

	HS [command-line options] <Enter>	
	HSM [command-	

	line options] <Enter >	
	HSA [comm and- line options] <Enter >	
	HSN [comm and- line	

	options] <Enter >	
	HSMN [comm and- line options] <Enter >	

	HSAN [command-line options] <Enter>	



The above batch files do the following:

1. first invoke and make memory-resident the Metagraphics graphics driver (first described and in Hypersignal-Macro Series Users Manual section 2.4, Selecting Graphics and Mouse Drivers)
2. run the appropriate Hypersignal software, applying optional command-line parameters if entered after the batch file name
3. upon exit from Hypersignal, remove the graphics driver from system memory, keeping the driver out of your system memory when Hypersignal is not in use

Upon executing Hypersignal software for the first time, a message of the form "cannot find configuration file...Initializing new configuration" should be displayed. This message only appears when executing the software for the first time, because normally the user-environment configuration save file "hsmacro.cnf" (or "hsacous.cnf") will be present as a result of performing an "exit" function after

a previous Hypersignal session. However, when running Hypersignal for the first time, no configuration file will be present, requiring that you to specify initial graphics and mouse driver selections.

For more information on Hypersignal configuration files, see Hypersignal-Macro Series Users Manual section 3.7, Updating and Accessing the User-Environment Configuration File, or Configuration Files in Hypersignal, below.

Please see Hypersignal-Macro Series Users Manual section 2.4, Selecting Graphics and Mouse Drivers, or Graphics and Mouse Driver Selection in Hypersignal User Mode below, for detailed information on making driver selections.

NOTE: If a configuration file is not found initially, then before printing the "...not found..." message Hypersignal first looks elsewhere, including the DOS path list, for a compatible configuration file. If one is found, then the software will start with this file, but save all new settings in a new file on the current path. This provides the means to set up a multiuser system with multiple configuration files: the initial configuration file can be provided from a master path, and individual, personalized configuration files saved to local paths. Also, since Hypersignal can be invoked from a distant path (see

note below), a multiuser system can be further enhanced to operate with only one copy of Hypersignal located on the master path.

You can invoke Hypersignal from a remote path. Generally, Hypersignal software searches for support files, its online Help file, and its printer-driver definition file first on its own path, then on the currently logged DOS path, and then in the DOS path list. The online Help and printer-driver definition files are also searched for on the path specified in the User Setup area (see Hypersignal-Macro Series User Manual section 3.3.3, The User Setup Area, or User Setup Area in Hypersignal, below).

Automatic detection and usage of an 80x87 math coprocessor is performed every time the software executes. For more details on this process, and overriding it if necessary, see Hypersignal-Macro Series Users Manual section 2.3, Using a Math Coprocessor, or Math Coprocessor Usage in Hypersignal, below.

Related Topics

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.1.1 Command-Line Options for Hypersignal

The following command-line options are available when executing Hypersignal-Macro series software packages:

Hypersignal Command-Line Options

Related Topics

Command-Line Options for Hypersignal

Command-Line Examples Of Running Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

EMS Memory Load Hypersignal Command-Line Option

[@j EMS Memory Load]

An '/M' entered immediately after the Hypersignal batch file name will cause Hypersignal to load its overlay file into EMS memory. However, this may limit the resolution of subsequent publication-quality prints--see Hypersignal-Macro Series Users Manual section 4.3.2, Printing Waveform Displays, or section 3.6 below, Printing Menus and Waveform Displays in Hypersignal User Mode.

Hypersignal Command-Line Options

EMS Memory Load

Evaluation Version DSP/Analog Base Address

Initial Macro Program

Initial Waveform Path

Engine Mode Switch

Evaluation Version DSP/Analog Base Address Hypersignal Command-Line Option

[@j Evaluation Version DSP/Analog Base Address]

For evaluation, university program packages, or other special versions of Hypersignal-Macro series software that require specific DSP/analog hardware to be installed in order to operate, a DSP/analog hardware address can be entered in the form '/$HHH', where HHH is a hexadecimal value.

Hypersignal Command-Line Options

EMS Memory Load

Evaluation Version DSP/Analog Base Address

Initial Macro Program

Initial Waveform Path

Engine Mode Switch

Initial Macro Program Hypersignal Command-Line Option

[@j Initial Macro Program]

An initial macro program filename can be entered in the form 'filename.mac'. If filename.mac specifies a valid macro source code file, then the software will immediately execute the macro file upon executing. This provides a method for executing Hypersignal functions remotely, from the

operating system or from another program. Input parameters with which the software will initialize the Macro Manager menu before executing the macro can be specified after the macro filename by preceding them with a '#' character. For example, you could instruct Hypersignal-Macro to run and then immediately compile (if necessary) and execute the "fsconv.mac" file included on distribution diskette #2 with the following command-line statement:

```
HSM fsconv #speech #9
#4 #sp_out  <Enter>
```

The above parameters for fsconv.mac would initialize the input waveform to "speech", the stretch and contract integers to 9 and 4, respectively, and the output waveform to "sp_out". Up to 6 input parameters can be placed in the command line. Note that the macro language file can use the "Exit" or "Abort" commands to automatically leave Hypersignal upon completion of the macro file, relinquishing control if necessary; see Hypersignal-Macro Series Users Manual section 7.10, Macro Commands and Functions, for more information. You could even specify an input parameter that the macro could use to *conditionally* determine whether to exit automatically, in case the macro was also one used frequently from inside Hypersignal, as in the following macro source code example:

```
if #3 = "Exit" then
```

```
    Exit   ; save config file and exit
endif

if #3 ="Abort" then
    Abort   ; exit without saving config file
endif
```

In the above example, Hypersignal will terminate depending on the value of input parameter #3 passed in the command line. However, if you needed to execute the macro from inside Hypersignal without exiting, input parameter #3 could entered in the Macro Manager menu as anything but "Exit" or "Abort" to prevent the macro from causing Hypersignal to terminate.

Hypersignal Command-Line Options

EMS Memory Load

Evaluation Version DSP/Analog Base Address

Initial Macro Program

Initial Waveform Path

Engine Mode Switch

Initial Waveform Path Hypersignal Command-Line Option

[@j Initial Waveform Path]

A '/pxxxx' entry in the Hypersignal command-line will set the initial Hypersignal waveform path to 'xxxx'. Normally, the waveform path is set using the User Setup function, under the Utilities menu, during program operation. An example is show below:

```
HSM
/pC:\hsmacro\waveform
```

Hypersignal Command-Line Options

EMS Memory Load

Evaluation Version DSP/Analog Base Address

Initial Macro Program

Initial Waveform Path

Engine Mode Switch

Engine Mode Switch Hypersignal Command-Line Option

[@j Engine Mode Switch]

A '/e' entry in the Hypersignal command-line will cause Hypersignal to run in "engine mode", in which case all user-interface graphics and menus, waveform displays, and other screen output is disabled. All functions, displays, and instruments are still active, but input must come from macro language programs or functions, or from the "simulated keyboard" macro control facility. In engine mode, the MetaWindow graphics drivers are not loaded into memory.

Engine mode can be especially useful when invoking Hypersignal from a user-defined C/C++ program. This technique is used by DSPower-Block Diagram software when controlling a Hypersignal package as a DSP Engine (see User Mode and Engine Mode, and Pipe.386 VxD, in Signalogic DSP

Software System Guide section 1.1, Relationship of Signalogic DSP Software Components).

Hypersignal Command-Line Options

EMS Memory Load

Evaluation Version DSP/Analog Base Address

Initial Macro Program

Initial Waveform Path

Engine Mode Switch

3.1.2 Command-Line Examples Of Running Hypersignal

Below is an example of invoking Hypersignal-Macro with both an initial macro program and initial waveform path:

```
HSM example.mac
/pc:\waveform  <Enter>
```

In the above example, the Hypersignal-Macro waveform pathname will be set to "c:\waveform", and the macro program "example.mac" will be executed upon Hypersignal-Macro startup.

Below is an example of invoking a university version of Hypersignal-Acoustic with an initial hardware base I/O address of 0x290:

```
HSA /$290  <Enter>
```

Below is an example of invoking Hypersignal-Macro EX from a

user-defined Windows C/C++ program:

```
WinExec("c:\\hsmacro\hsmacro.exe /e /pc:\\waveform acq8chan.mac",
SW_HIDE);
```

Note that in the above example the associated batch file (which would have been hsm.bat) is not used, because otherwise the MetaWindow graphics drivers would be loaded. Note also that the engine mode parameter is specified, which diables all forms of Hypersignal video, mouse, and physical keyboard interaction.

Related Topics

Command-Line Options for Hypersignal

Command-Line Examples Of Running Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

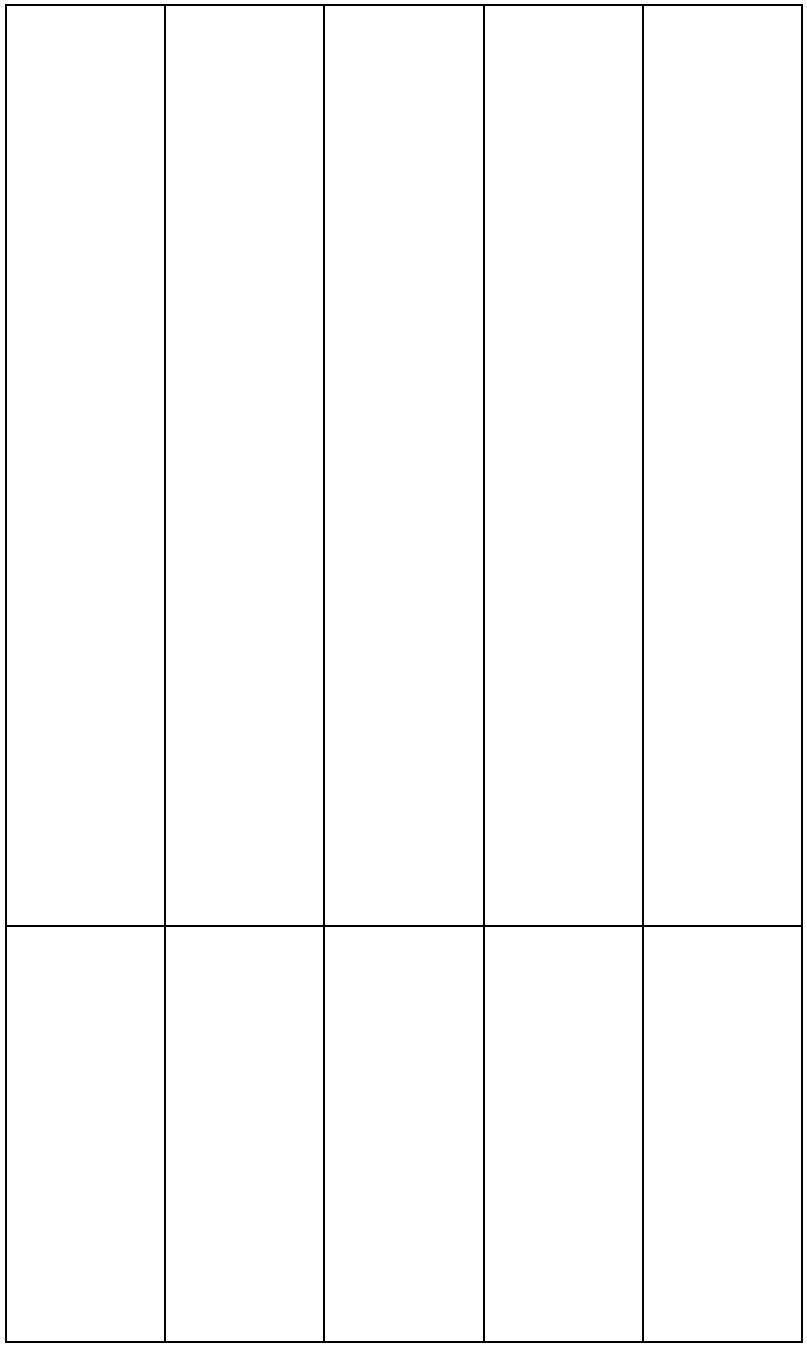
3.2 Keyboard and Mouse Interface In Hypersignal User Mode

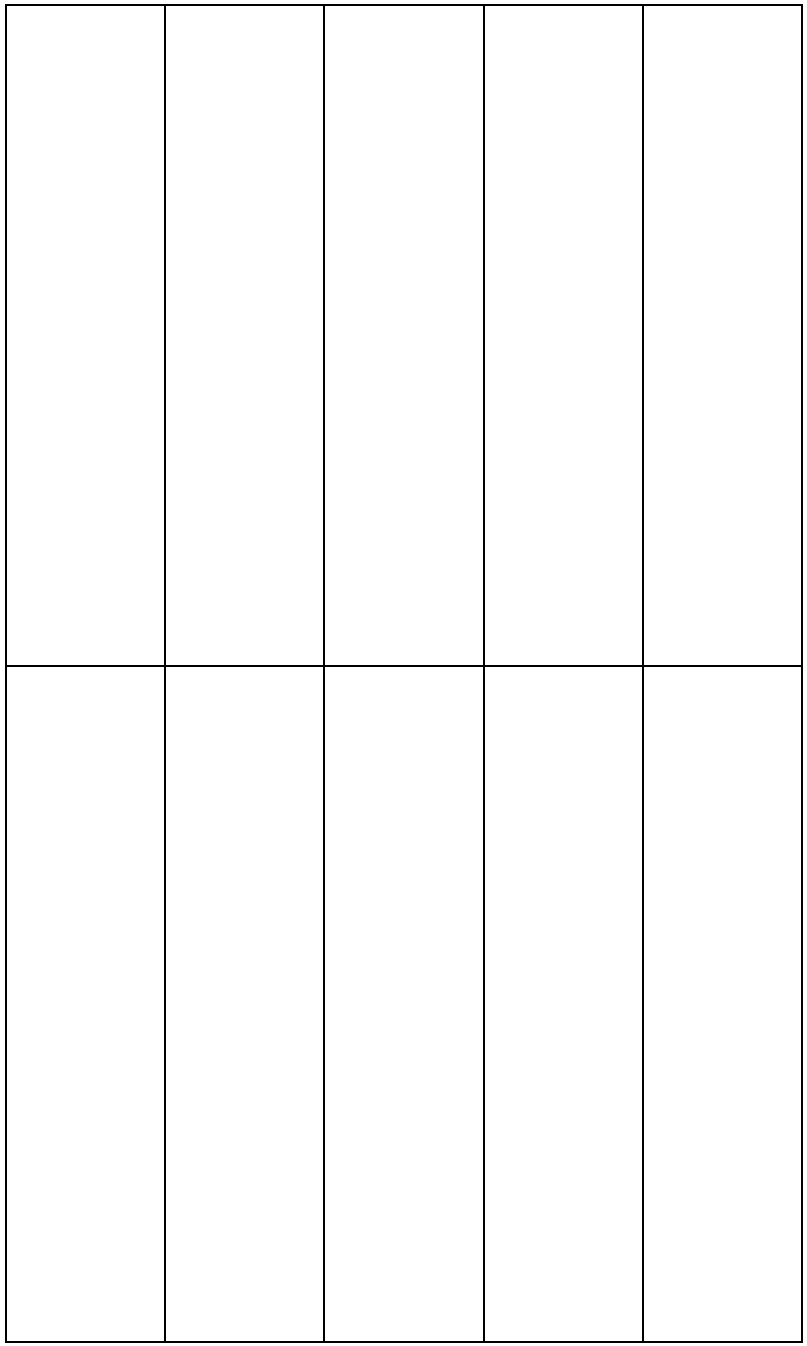
NOTE: If a Hypersignal-Macro series software package is being operated in engine mode, then the interface below still applies, except that data must be received through the "simulated keyboard" macro facility. See User Mode and Engine Mode, in Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components.

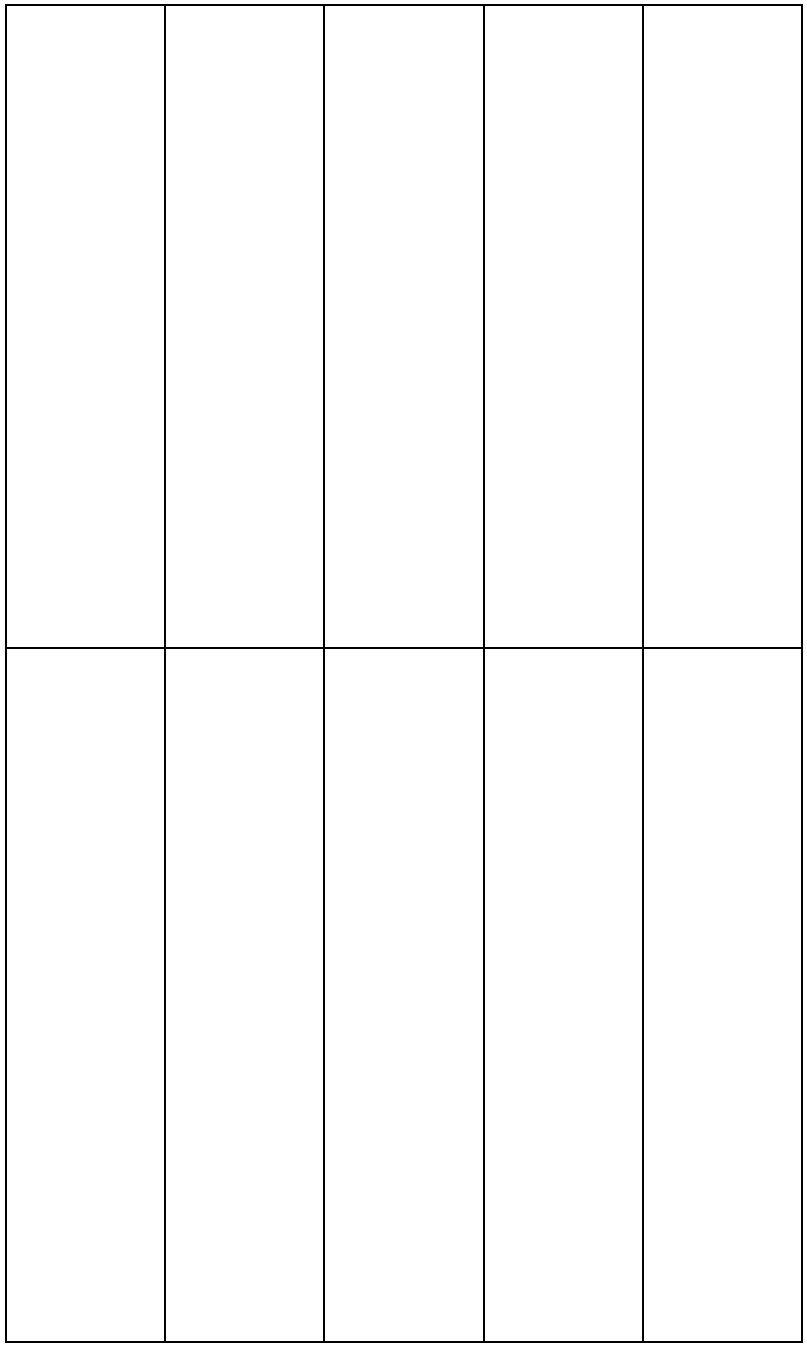
Below is listed the general function of keyboard keys and mouse buttons during the four basic types of Hypersignal displays: main menu, data-entry menu, waveform display, and online Help display. Only commands that are used frequently in all four major areas are listed here. For a complete list of data-entry editor, waveform display, and online Help commands, see Hypersignal-Macro Series Users Manual sections 3.4, 4.3, and 3.2, respectively, or sections 3.5, 3.3, and 3.13 (Data-Entry Editor in Hypersignal User Mode, Interactive Waveform Displays in Hypersignal User Mode, and Online Help in Hypersignal User Mode), respectively. Note that for two-button mice, functions listed as responding to the middle button,

"two leftmost" buttons, or "two rightmost" buttons will be deactivated. For more detailed information on individual keyboard and mouse functions, please see the appropriate section in this manual pertaining to the display/DSP function you are concerned with.

Table 2General Hypersignal Keyboard Interface in User Mode







General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode
Main Menu in Hypersignal User Mode
Data-Entry Editor in Hypersignal User Mode
Printing Menus and Waveform Displays in Hypersignal User Mode
Running External Programs in Hypersignal
Configuration Files in Hypersignal
Creating Waveform Files in Hypersignal
Finding Waveforms in Hypersignal User Mode
Running Macro Programs in Hypersignal

3.3 Online Help In Hypersignal User Mode

Online Help may be accessed from anywhere in the Hypersignal program by using the 'F1' key, or the rightmost two buttons on a three-button mouse. From the main menu and inside waveform displays, the '?' and 'H' keys also access online Help. During waveform displays, always use these keys and/or mouse bottoms, even though legends may not always be shown during waveform displays.

Within any Help text display, accessing Help again will begin a general Help text display; the general Help text is always available even if the online Help file ("hsmacro.hlp") should be lost or destroyed.

The number of pages in the Help display will vary between menu

selections. Use the 'PgDn', down arrow keys, or move the mouse down to page forward, and the 'PgUp', up arrow keys, or move the mouse up to page backward. The 'End' key (or righthand mouse button) will jump to the end of the displayed Help text. The 'Home' key (or lefthand mouse button) will jump to the start of the displayed Help text. You may return to the previous menu at any time by pressing the 'Esc' key (or the leftmost two buttons on a three-button mouse).

Hypersignal searches the currently logged DOS path, the path containing the "HSxxxxx.EXE" file (DOS 3.3 and above only), the User Setup path, and the DOS path list for the online Help file. If not found, Hypersignal then searches the current subdirectory and logged drive shown in the User Setup area. If still not found, Hypersignal will report the Help file not found and disable online Help display during the current session until such time as the current subdirectory and logged drive in the user setup area is changed to correctly access the online Help file. General Help is always available, since it is imbedded in the main program file, "HSxxxxx.EXE". If you should accidentally damage or destroy the online Help file, then contact your distributor for a replacement.

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.4 Main Menu in Hypersignal User Mode

After starting the Hypersignal program as outlined in previous sections, the Hypersignal main menu similar to one will be displayed. Once the main menu is displayed, the following keys are active (note that most of these key functions are listed horizontally at the bottom of the main menu screen):

Table 3 Main Menu Keyboard Interface in Hypersignal

	Quits Hypersignal without saving or updating the user-environment configuration file
	Exits Hypersignal and saves/updates the user-environment configuration file
	Saves/updates the user-environment configuration file

without exiting Hypersignal

Begins online Help display for current menu function selected (highlighted selection)

Invokes the Macro Manager menu, in which a macro source file can be specified for compilation and/or execution (see Hypersignal-Macro Series Users Manual section 3.10, Invoking Macros, or section 3.12, Invoking Macro Programs in Hypersignal User Mode)

Invokes the Macro Record menu, in which an empty macro source file can be specified to which to record manual function and display selections (and display commands) for later "playback" as a macro (see Hypersignal-Macro Series Users Manual section 4.10.2, Using the "Record" Mode to Create Macros, or Recording Macros in Hypersignal User Mode in section 3.22, Creating Macro Programs in Hypersignal User Mode).

This is the DOS "Hot-Key"--it activates a temporary DOS shell in which standard DOS commands can be entered and executed (type 'Exit' to return to Hypersignal)

Change current (highlighted) menu selection in respective direction (up, down, left, or right)

Selects currently highlighted menu function

The main menu can be divided into three basic areas. The largest, occupying the lower 2/3 of the screen, is the menu selection area. In this area, the arrow keys or mouse can be used to change the current menu highlight, and the 'Enter' (carriage return) key or any mouse button can be used to select a particular function.

At the upper right of the screen is the User Setup area. This area may be accessed by the User Setup menu selection, under the Utilities column heading, at the far right of menu selection area. In this area, the current logged drive and subdirectory, currently selected time and frequency domain filenames, and several global program parameters may be modified. See section 3.4.3, User Setup Area in Hypersignal, for more detailed information on this part of the Hypersignal main menu.

At the upper left of the screen is the System Status area. This area cannot be accessed by the user. Status parameters include the current time and date, running Hypersignal session time, and current menu function selected.

The current menu function selected will display "MAIN MENU" until menu selection, after which it will display the appropriate menu label. The System Status area is also used by the data-entry editor as a "Waveform Summary" display area (see Hypersignal-Macro Series Users Manual section 3.4.1, Waveform Summary Display, or section 3.5, Waveform Summary Display in Hypersignal) when waveforms have been loaded, and for a "Popup Waveform Directory" display area when the program cannot locate a specified waveform filename (see Hypersignal-Macro Series Users Manual section 3.4.2, Popup Directory Display, or section 3.5.2, Popup Directory Display in Hypersignal).

See Hypersignal-Macro Series Users Manual section 3.3.4, The System Status Area, or section 3.4.4, System Status Area in Hypersignal, for detailed information on the System Status area of the main menu.

Related Topics

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.4.1 Menu Selection in Hypersignal User Mode

When being operated in user mode, all Hypersignal functions can be highlighted in the main menu using the mouse motion or arrow keys and then selected with the 'Enter' (carriage return) key or any mouse button. There are three categories of Hypersignal functions, which are divided into columns in the main menu display: Time Domain, Frequency Domain, and Utilities. Appendix B, Hypersignal Environment Flow Diagram, shows a graphical representation of all menu selections and waveform displays, and what type of waveform data they use as input and create as output, if any. Pressing the 'Enter' key, space bar, or any mouse button will invoke the currently highlighted menu selection.

Once inside the "data-entry menu" for any menu selection (see Hypersignal-Macro Series Users Manual section 3.4, Using the Hypersignal Data-Entry Editor, or section 3.5, Data-Entry Editor in

Hypersignal User Mode), pressing the 'Enter' key (or any mouse button) with the cursor on the last data-entry field, or pressing the 'ctrl-Enter' key sequence at any time advances to function processing or graphical display, as the case may be.

At any time, you may access online Help for the currently highlighted menu selection. Use the '?', or 'F1' keys to display of relevant Help information. *The 'Esc' (escape) key always returns to the main menu.*

Related Topics

Menu Selection in Hypersignal User Mode

Quit/Exit in Hypersignal User Mode

User Setup Area in Hypersignal

System Status Area in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.4.2 Quit/Exit in Hypersignal User Mode

In user mode, you can only leave Hypersignal from the main menu. You can do this two ways: Quit ('Q' key), or Exit ('E' key). If you quit, Hypersignal stops everything on the spot, the current Hypersignal user-environment is not saved, and the operating system prompt returns immediately. If you exit, Hypersignal saves the current user environment, including graphics and acquisition hardware driver selections, all current filename selections, display options, menu selections, and so forth, in the user-environment configuration file ("hsmacro.cnf" or "hsacous.cnf") and then returns to the operating system prompt.

When you leave Hypersignal, the current DOS path will be the same as when you started Hypersignal, unless you shelled out to DOS or to an external program during Hypersignal execution and changed the logged path with the RESTORE DOS PATH field in the External Program Configuration menu disabled (see Hypersignal-Macro Series Users Manual section 3.6, Invoking External Programs and the DOS "Hot-Key", or section 3.7, Running External Programs in Hypersignal).

Related Topics

Menu Selection in Hypersignal User Mode

Quit/Exit in Hypersignal User Mode

User Setup Area in Hypersignal

System Status Area in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.4.3 User Setup Area in Hypersignal

In user mode, when menus are displayed (screens other than graphical displays), the upper right of the screen will reflect the current User Setup. This area may be accessed by the User Setup menu selection, under the Utilities column heading (see Hypersignal-Macro Series Users Manual section 2.5.1, Initializing the User Setup Area). In this area, such parameters as the current logged drive and subdirectory, currently selected time and frequency domain waveform filenames, dual

display autoscaling and global program sound enable/disable may be modified.

In engine mode, some types of data normally displayed in the User Setup area are sent via the Pipe.386 VxD to the external controlling application, which can choose to process or ignore this data.

The logged drive and subdirectory parameters *apply to all Hypersignal waveform files.* The current TIME FILENAME and FREQ FILENAME can be used to initialize the functions under the Time Domain and Frequency Domain column headings, respectively. In practice, setting these values via the User Setup menu item is of little worth, because all filenames can be set in the individual functions anyway, and the program does such a good job of automatically tracking waveform names from one function to another.

All user setup parameters are initialized from the user-environment configuration file, including the current drive and subdirectory. If a configuration file cannot be found, then the User Setup parameters will be initialized to a default state (drive and subdirectory will be initialized to their DOS values at the time of program start).

Should subdirectory information need be entered that is longer than the subdirectory field in the system setup area, additional subdirectory

information may be entered following the drive identifier (drive plus colon) in the drive field. When accessing files, Hypersignal forms the search path by concatenating the drive and subdirectory fields, *exactly as they are shown onscreen in the User Setup area.* If using both fields for subdirectory information, be careful that consecutive or extra forward slashes do not occur.

If you need to specify a path longer than the DRIVE and SUBDIRECTORY fields combined, then set this path directly in the user-environment configuration file. Even though the path will be truncated for display purposes on the main menu, its internal length will remain intact. In such a case, do not edit the path directly in the program. If you do, its internal value will be truncated also.

Additional Information

Related Topics

Menu Selection in Hypersignal User Mode

Quit/Exit in Hypersignal User Mode

User Setup Area in Hypersignal

System Status Area in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

Autoscaling in Hypersignal

The DUAL AUTOSCALE field in the User Setup area is important enough to warrant further discussion in this section. This field disables or enables automatic scaling by the program during dual waveform displays. When enabled (an entry of 'ON'), the program automatically scales the smaller of two waveforms in a dual display to so that its maximum absolute amplitude matches the larger. When disabled (an entry of 'OFF'), no scaling is performed, allowing waveform amplitudes to be compared relatively. The default value for this field is 'FILE', in which case the program automatically enables autoscaling for dual displays where one or both waveforms have the "AUTOSCALE" bit in their waveform headers enabled, and disables autoscaling in all other cases. Hypersignal does not

currently set the AUTOSCALE bit for any reason, but this bit can easily be set by user-defined external programs or macro language processing. See Hypersignal-Macro Series Users Manual chapter 5, Hypersignal Waveform File Format, or Signalogic DSP Software Reference Guide section 4.1, Hypersignal Waveform File Format, for more information about the AUTOSCALE bit and waveform headers.

Related Topics

Menu Selection in Hypersignal User Mode

Quit/Exit in Hypersignal User Mode

User Setup Area in Hypersignal

System Status Area in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.4.4 System Status Area in Hypersignal

In user mode, when menus or data-entry screens are displayed (screens other than graphical displays), the upper left of the screen shows the "System Status". This area cannot be accessed by the user. System Status area parameters include the time and date, running Hypersignal session time, and current function selected.

In engine mode, some types of data normally displayed in the System Status area are sent via the Pipe.386 VxD to the external controlling application, which can choose to process or ignore this data.

The current function selected will display "MAIN MENU" while the main menu is displayed, and indicate the current function after selection from the main menu.

The time and date are initialized at program start, and updated with normal program use thereafter.

The running session time indicates the length of time Hypersignal has been in use continuously. It is also updated with normal program use. If no clock is active in the host computer, then the time and date will be correct only if they were set manually at the time of the DOS prompt during computer power-on sequence.

The System Status area is also used to display various other information

during the course of normal program operation:

Additional Information

Related Topics

Menu Selection in Hypersignal User Mode

Quit/Exit in Hypersignal User Mode

User Setup Area in Hypersignal

System Status Area in Hypersignal

Error Message Display in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

Status Message Display Area in Hypersignal

Various Hypersignal status message and non-critical error conditions may be reported in the

System Status area, located at the upper left of the Hypersignal main menu, from time to time. See Hypersignal-Macro Series Users Manual section 3.3.4.1, Status Message Display, or Status Message Display in Hypersignal, below, for more information.

Related Topics

Status Message Display Area in Hypersignal

Error Message Display Area in Hypersignal

Waveform Summary Display Area in Hypersignal

Popup Waveform Directory Display Area in Hypersignal

Status Message Display in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

Error Message Display Area in Hypersignal

Various DOS and Hypersignal critical error conditions may be reported in the System Status area, located at the upper left of the Hypersignal main menu, from time to time. See Hypersignal-Macro Series Users Manual section 3.3.4.2, Critical Error Message Display, or Error Message Display in Hypersignal, below, for more information.

Related Topics

Status Message Display Area in Hypersignal

Error Message Display Area in Hypersignal

Waveform Summary Display Area in Hypersignal

Popup Waveform Directory Display Area in Hypersignal

Status Message Display in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Waveform Summary Display Area in Hypersignal

Upon successful load of a waveform for subsequent display or processing, a summary of the contents of waveform file headers is displayed in the System Status Area, located at the upper left of the Hypersignal main menu. See section Hypersignal-Macro Series Users Manual section 3.4.1, Waveform Summary Display, or Waveform Summary Display in Hypersignal, below, for details.

Related Topics

Status Message Display Area in Hypersignal

Error Message Display Area in Hypersignal

Waveform Summary Display Area in Hypersignal

Popup Waveform Directory Display Area in Hypersignal

Status Message Display in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

Popup Waveform Directory Display Area in Hypersignal

When a specified waveform filename can't be located, or upon user request, the System Status area (located at upper left of the Hypersignal main menu) is used to show popup directory information based on the current processing/display function being executed. A waveform filename search strategy is employed to allow the user to search through likely categories of waveforms depending on what the required waveform file types are in the current function. See Hypersignal-Macro Series Users Manual section 3.4.2, Popup Directory Display, or Popup Directory Display in Hypersignal, below, for a detailed description.

Related Topics

Status Message Display Area in Hypersignal

Error Message Display Area in Hypersignal

Waveform Summary Display Area in Hypersignal

Popup Waveform Directory Display Area in Hypersignal

Status Message Display in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

Status Message Display in Hypersignal

In user mode, non-critical Hypersignal-detected error or status conditions, such as a user-defined program not found or lack of disk space to perform an "Exit" function (and save the user-environment configuration file) are displayed in the System Status area. Also, the macro language facility uses this area extensively, displaying all compilation and execution status or error messages in the System Status area.

In engine mode, status and non-critical error message display can also appear as a status box in the external controlling application's user-interface, if the external application accepts this data being sent via the Pipe.386 VxD. DSPower-Block Diagram software has an option to enable a status and

error message box when a Hypersignal block is being executed..

Related Topics

Status Message Display Area in Hypersignal

Error Message Display Area in Hypersignal

Waveform Summary Display Area in Hypersignal

Popup Waveform Directory Display Area in Hypersignal

Status Message Display in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

Error Message Display in Hypersignal

Critical errors, including DOS-related error conditions such as write-protected diskettes, drive not ready, and media error (bad sector, seek error, etc.) are displayed in the System Status area at the upper left

of the main menu. All DOS error messages that would normally overwrite the current display (including the infamous "Abort, Retry, or Ignore?" messages) are intercepted by Hypersignal, translated, and displayed (if necessary) in the System Status area along with the disk affected, if any.

In engine mode, critical error message display can also appear as a status box in the external controlling application's user-interface, if the external application accepts this data being sent via the Pipe.386 VxD. DSPower-Block Diagram software has an option to enable a status and error message display box when a Hypersignal block is being executed..

Related Topics

Status Message Display Area in Hypersignal

Error Message Display Area in Hypersignal

Waveform Summary Display Area in Hypersignal

Popup Waveform Directory Display Area in Hypersignal

Status Message Display in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.5 Data-Entry Editor in Hypersignal User Mode

NOTE: The Hypersignal data-entry editor is intended for user mode operation. However, interactive commands can be sent to the editor from macro programs, and from an external controlling application using the Pipe.386 VxD.

The Hypersignal data-entry editor is used to allow convenient entry and modification of parameters needed to execute Hypersignal processing and display functions.

Modifiable screen data for all data-entry menus are arranged as a column of "fields". A field is a specific place on the screen where data may be entered or changed. Each field consists of one or more characters. Characters may be any valid keyboard character except for the spacebar, unless otherwise noted. The current character position in the current field being edited is indicated by a blinking

cursor. A large blinking cursor is used when in insert mode (enabled by the 'Insert' key; see below), and the standard DOS cursor is used when in Overwrite mode. A field is "entered" by the 'Enter' (carriage return) key or a mouse button, after which the cursor advances to the next field automatically. The arrow keys or mouse motion can be used to move the cursor from character to character and from field to field.

There are two ways to advance to actual function processing or graphical display:

1. entering the last field (pressing the 'Enter' key or any mouse button when the cursor is on the last field in the menu), or
2. pressing the <ctrl-Enter> key sequence (use <ctrl-J> key sequence on NEC PC-98xx machines). Doing either of these will cause function processing or display to begin, as appropriate.

All fields in a data-entry menu except the last field may be revised as many times as necessary by repositioning the cursor to that field and entering the field.

At any time, you may access online Help for the currently invoked processing or display function. Use the 'F1' key to begin display of relevant Help information.

To return to the main menu from any data-entry menu, use the 'Esc' (escape) key.

The data-entry editor has two basic modes, controlled through the EDITOR UNDO field in the User Setup menu selection. The default value for this field is 'OFF', which will cause the up and down arrows or vertical mouse motion to keep

any changes made in the previous field. An 'ON' entry for the EDITOR UNDO field will cause the up and down arrow keys and vertical mouse motion to "undo" any changes on the previous field--allowing only the 'Enter' key or mouse button to finalize a field.

When a filename field is entered, data beyond the cursor position is cleared. This does not occur for fields which contain "subfields", or two or more separate entries separated by commas, since it may only be one subfield of the field that needed changing. This also does not occur in almost all cases if the cursor is at the very beginning of a field, regardless of the type of field.

After entering the Insert mode (see 'Insert' key, below), any type of entry that causes a field change (up and down arrow keys, mouse motion, 'Enter' key) automatically disables the Insert mode.

Following are the active keyboard controls when editing Hypersignal function data entry displays:

Table 4Data-Entry Editor Interface in Hypersignal

	All arrow keys and mouse movement are active in the editor, allowing you to move within and between fields without changing data. Arrow keys will "wrap-around" when at the first and last fields up and down, or the first and last characters left and right. Mouse motion never causes a wrap-around, because it is so easy to overmove the mouse.

Used to "enter" a field. Whatever data presently contained in a field will be finalized when the 'Enter' key or a mouse button is pressed. A field may be entered regardless of the current cursor position in the field. After entering the last field in a data-entry menu, function processing or graphical display, as appropriate, will begin.

"Enters" the current field and causes function processing or graphical display to begin.

Deletes the character in the current cursor position without changing cursor position. All characters to the right of the cursor are moved to the left one space.

Causes the data-entry editor to enter Insert mode, in which new characters do not overwrite characters already entered, but instead are inserted at the current cursor location. Insert mode is indicated onscreen by a large blinking cursor. Pressing the 'Insert' key again returns to Overwrite mode, and also returns the cursor to its

original, standard DOS size.

Deletes the character immediately to the left of the current cursor position, and then moves to the left one space. All characters to the right of the cursor are moved to the left one space.

Returns to the main menu. Any fields already entered remain so.

Begins popup directory display for the current function, if not already started because the program couldn't locate a specified waveform filename. The waveform filename and extension used depends on the current function and/or menu selection. Consecutive use of the 'PgUp' key causes the popup directory display to first return to the first page of the current directory display (if not on the first page), and then starts changing the current filename and extension search strategy in use. See Hypersignal-Macro Series Users Manual section 3.4.2, Popup Directory Display, or Popup Directory Display in Hypersignal, below, for details.

Used to display the next page of popup directory information. Popup directory information is displayed during functions requiring input waveform filename entry when the input filename cannot be found, or in response to the 'PgUp' key, above. See Hypersignal-Macro Series Users Manual section 3.4.2, Popup Directory Display, or Popup Directory Display in Hypersignal, below, for details.

Moves cursor to beginning of current field.

Moves cursor to end of current field.

Causes online Help text for the currently invoked processing or display function to be displayed. The 'Esc'

(escape) key will return
to the data-entry menu.

Related Topics

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.5.1 Waveform Summary Display in Hypersignal

In user mode, waveform summaries are displayed in the System Status area of the Hypersignal main menu (upper left) after successful loading of waveform file(s) for display or processing. A waveform summary displays parameters and attributes from the waveform file header (see Hypersignal-Macro Series Users Manual chapter 5, Hypersignal Waveform File Format, or Signalogic DSP Software Reference Guide section 4.1, Hypersignal Waveform File Format), including the number of samples in the waveform, type of data (arbitrary time waveform, LPC autocorrelated, filter construction output, etc.), whether or not the time domain file is real or complex, number of channels, maximum amplitude, sampling frequency, FFT order, and time domain analysis framesize.

In engine mode, waveform summary display can also appear as

a status box in the external controlling application's user-interface, if the external application accepts this data being sent via the Pipe.386 VxD. DSPower-Block Diagram software has an option to enable a waveform summary display box when a Hypersignal block is being executed..

To redisplay a waveform summary that has been overwritten by popup directory display information or error messages, you can reload the waveform by repositioning the cursor at the beginning of the waveform filename field and pressing the 'Enter' (carriage return) key.

In the case of dual-waveform display functions, the waveform summary always applies to FILENAME1.

Below is an example of a waveform display summary:

Especially note the "Type/complex/numchn" descriptor, which, in this example, indicates a non-complex, single-channel, filter (transfer function, or "H[n]") magnitude (frequency domain) waveform file. The "type" of a waveform actually consists of two parts: the type of data, and its original source. Currently, the type of waveform data can be "tim" for time domain data, "mag" for frequency domain magnitude data, "phz" for frequency domain phase data, or "pwr" for frequency domain power spectra data. The source of data can be "" (null) for arbitrary time domain data (e.g. A/D input), "H[n]" for filter construction or other transfer function output, or "LPC" for LPC Autocorrelation output. The "complex" descriptor can be "real" for data that was non-complex in its original time domain form, or "cplx" for data that was complex. Complex data exists when at least one imaginary component in a time waveform file is non-zero. See Hypersignal-Macro Series Users Manual chapter 5, Hypersignal Waveform File Format, for a description of the complex attribute of waveform file headers and the format of both complex and real data.

Below is a second example of a waveform display summary:

The above example is for a complex, two-channel, arbitrary time domain waveform file.

NOTE: With version 3.10 or later of Hypersignal-Macro, the "Max |amplitude" value that is displayed can be a floating-point value. For a full explanation of waveform amplitude values and waveform header exponents, see Hypersignal-Macro Series Users Manual chapter 5, Hypersignal Waveform File Format, or Signalogic DSP Software Reference Guide section 4.1, Hypersignal Waveform File Format.

Related Topics

Waveform Summary Display in Hypersignal

Popup Directory Display in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.5.2 Popup Directory Display in Hypersignal

In user mode, popup directory display appears during data-entry editor menus as an automatic window display in the System Status area (upper left of main menu) when a specified filename cannot be found. Also, you can manually invoke popup directory display from inside any data-entry menu by using the 'PgUp' key.

In engine mode, popup directory display can also appear as a status box in the external controlling application's user-interface, if the external application accepts this data being sent via the Pipe.386 VxD. DSPower-Block Diagram software has an option to enable a

popup waveform directory box when a Hypersignal block is being executed..

Popup directory display includes waveform filename and extension, length, and date of creation. In user mode, the search strategy for different waveform extensions and/or more specific waveform filename may be changed via consecutive use of the 'PgUp' key (see Hypersignal-Macro Series Users Manual section 3.4.2.1, Popup Directory Waveform File Search Strategy, or Popup Directory Waveform File Search Strategy in Hypersignal User Mode, below).

During time domain, frequency domain, and some utility functions requiring input waveform files, if a waveform filename is not found after being entered by the user, then candidate filenames having appropriate filename extensions will "popup" in the System Status area. The initial filename extension used is based on the current function. For example, when performing an FFT function, an input filename not found will prompt display of those filenames on the current path with ".TIM" extensions, since the FFT is a time domain function. Other time domain extensions include ".FFD" and ".IFD" for Filter Construction filenames, ".IIR" for Recursive Filtering input filter coefficient files, and ".EQU" for Difference Equation save files. Continued use of the PgUp key will change the

search path to "alternate" filename extensions, or a "filename.*" search (see Hypersignal-Macro Series Users Manual section 3.4.2.1, Popup Directory Waveform File Search Strategy, or Popup Directory Waveform File Search Strategy in Hypersignal User Mode, below).

During frequency domain functions requiring filename entry initial filename extensions used for popup directory display can be ".FRQ" (magnitude), ".PHZ" (phase), ".IMA" (imaginary), ".REA" (real), or ".PWR" (power spectra).

For utilities selections the value for the extension varies, depending on menu selection. For External Program Configuration, the extensions are ".COM", ".EXE", and ".BAT". For ASCII Export, the extension is ".TXT". For File Acquisition and ASCII import, the extension used is the extension currently entered as part of the input filename field of the data-entry menu, or the last valid one set for time domain or frequency domain functions. This is also true for all other menu selections that do not require filename entry: they use the last valid extension set by selections that do require filename entry.

Additional Information

Related Topics

Waveform Summary Display in Hypersignal

Popup Directory Display in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

Popup Directory Waveform File Search Strategy in Hypersignal User Mode

When Hypersignal is being operated in user mode, consecutive use of the 'PgUp' key will change the search strategy for the current popup directory display. After initial display, in which the entered filename has been replaced by a "*" and an appropriate filename extension added, a second 'PgUp' key will cause the currently entered waveform filename and extension to be used in the directory search. If no extension is entered, a ".*" extension will be added to the entered filename. Thus, the second 'PgUp' key effectively shows all

occurrences of the filename you've entered, regardless of extension. A third 'PgUp' key will use the currently entered filename along with an "alternate extension"; for example, filter design functions have a primary extension of ".IFD" or ".FFD" because of the specification files they use, but they also have an alternate extension of ".TIM", because they are time domain functions. As another example, Difference Equations have a ".EQU" primary extension and a ".TIM" alternate extension. If no alternate extension applies to the function selected, a ".*" extension is substituted. Further 'PgUp' keys cause the process to start over.

You may also temporarily enter "*" or "?" wildcard characters after or inside the filename to further direct a waveform search; for example a 'test*' entry would include all filenames starting with "test", and having the appropriate extension. As another example, an entry of 'x?test' would include all filenames starting with "x", ending with "test", and having any other character in between.

NOTE: To re-initialize popup directory display after you have changed the filename in the data-entry menu, or added a wildcard character ('*', '?'), >you must reposition the cursor at the beginning of the waveform filename field and 'Enter' the field, then press the 'PgUp' key.

Related Topics

Waveform Summary Display in Hypersignal

Popup Directory Display in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.6 Printing Menus and Waveform Displays in Hypersignal User Mode

To print Hypersignal menu displays in user mode, you should use the standard DOS <shift-Print Scrn> key sequence; there is no built-in Hypersignal function for printing menu displays.

To print waveform displays, both to printers and disk files, you should use the built-in Hypersignal

function that can perform both "screen dumps" and "publication-quality" prints. A screen dump is defined as simply transferring the current contents of video RAM directly to the printer, without any increase in resolution. Of course, for printers with resolution greater than about 90 dots per inch, this will not fully utilize the capability of the printer. A publication-quality print is defined as regenerating the current screen display in EMS memory, at much higher resolution, and then transferring this image to the printer.

The built-in waveform print function is invoked by the 'F2' key inside all waveform displays. See Hypersignal-Macro Series Users Manual section 4.3.2, Printing Waveform Displays, or Printing Menus and Waveform Displays in Hypersignal User Mode, below, for full details on using the built-in print function.

NOTE: If you need to add a printer to the printer-driver definition table (pdt), or modify an existing entry, then you can invoke the pdt editor. The file "BDE.EXE" has been provided for this purpose; by typing 'BDE' at the DOS command line (outside of Hypersignal), you will bring up a listing of all current printers. A special-purpose editing mode is available that allows you to set various printer specifications and parameters, and also to set special "escape sequence" commands for the printer. Follow

the instructions in the online Help for when modifying or adding printer entries. If you need further assistance, please contact your distributor.

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.7 Running External Programs in Hypersignal

When operating in user mode, external programs can be executed directly from the Hypersignal main menu. External programs can also be executed from Hypersignal macro programs, when Hypersignal is operating in engine mode.

To enhance the main menu capability, "User-Defined" menu labels are available on the main

menu (two each under the Time Domain, Frequency Domain, and Utilities columns). For example, you could set up a user-defined utility menu label to point to a text-editor, and time domain and frequency domain menu labels to point to your own DSP processing programs.

In user mode, select the Ext Program Config menu item to set up your external program filenames and desired Hypersignal menu labels. For example, a typical External Program Configuration menu might look like this:

where "sinegen.exe" is an example external program provided on the distribution diskettes (intended as an example of how to access the Hypersignal user-environment configuration file), "edit.com" is a prototypical text editor, and "progN.ext" are prototypical user-defined programs.

In user mode, it is quite possible there will not be enough memory available to run an external program from Hypersignal. In this case, see if you reduce the number of memory-resident utilities loaded or relocate them to high memory. Also, you may investigate whether you can load the "met_shel.exe" graphics driver into "high" memory, above the 640 kByte conventional memory. Whether this can be done or not is highly system-dependent. See Hypersignal-Macro Series Users Manual section 2.4.3, Installing the Metagraphics Driver Above Conventional Memory, for more information.

In engine mode, it is unlikely that external programs will encounter memory problems. First, the met_shel.exe graphics driver is not loaded in engine mode; second, external programs can actually be executed by the external controlling application, after being passed an external program pathname and command-line parameters through the Pipe.386 VxD. This is the method used by macro programs and functions auto-generated by DSPower-Block Diagram for

Hypersignal when being used as a DSP Engine. Also, macro programs can ask the controlling application to execute external programs either synchronously, or aysnchronously.

NOTE: In user mode, you should not install memory-resident drivers from Hypersignal. Why? Because of the graphics driver ("met_shel.exe"), which is installed as memory-resident just before Hypersignal runs (done automatically by the HSx.BAT batch file), and is removed from memory just after Hypersignal runs. In order for "met_shel.exe" to be removed, it must be the last memory-resident driver installed. If not, because another driver has been installed after it, "met_shel.exe" cannot be removed and will continue to occupy system memory even after Hypersignal has finished executing.

See Hypersignal-Macro Series Users Manual section 2.5.3, Initializing the External Program Configuration Menu, for more information on setting up external programs and defining main menu labels.

Related Topics

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode
Main Menu in Hypersignal User Mode
Data-Entry Editor in Hypersignal User Mode
Printing Menus and Waveform Displays in Hypersignal User Mode
Running External Programs in Hypersignal
Configuration Files in Hypersignal
Creating Waveform Files in Hypersignal
Finding Waveforms in Hypersignal User Mode
Running Macro Programs in Hypersignal

3.7.1 DOS Shell in Hypersignal User Mode

The F10 key has been installed in the Hypersignal main menu as a DOS "Hot-Key", which allows you to jump to a DOS shell at any time from the main menu, perform DOS operations, and "exit" quickly back to Hypersignal.

A DOS shell is invoked with the same technique used for other external programs, with the external program in this case being a copy of the standard DOS "COMMAND.COM" file.

NOTE: Hypersignal updates the user-environment configuration file before executing an external program (including DOS shell) and reads the configuration file back again after the external program finishes executing. In this way, a user-created external program can access and modify the

configuration file, causing changes in the Hypersignal environment upon return. In order to guarantee that the configuration file can be read after the external program (or DOS shell) returns, Hypersignal saves the current DOS path (current logged drive and/or subdirectory) before yielding control, and restores the saved value afterwards. If you wish to override the automatic restoration, because external or macro processing needs to change DOS paths, you can set the RESTORE DOS PATH field in the External Program Configuration menu to 'N'. However, keep in mind that if you disable automatic restoration, and then change the current logged drive and/or subdirectory while in an external program or DOS shell, Hypersignal will read (or attempt to read) a new configuration file, if any, after the external program finishes.

You should not install memory-resident drivers from a DOS shell invoked during Hypersignal operation. Why? Because of the graphics driver ("met_shel.exe"), which is installed as memory-resident just before Hypersignal runs (done automatically by the HSx.BAT batch file), and is removed from memory just after Hypersignal runs. In order for "met_shel.exe" to be removed, it must be the last memory-resident driver installed. If not, because another driver has been installed after it, "met_shel.exe" cannot be

removed and will continue to occupy system memory even after Hypersignal has finished executing.

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.8 Configuration Files in Hypersignal

The "user-environment configuration file" is an ASCII text file that contains the complete state of all user-controlled parameters in Hypersignal. This file is intended to be saved between Hypersignal sessions so that frequent use of the software is as convenient as possible, and no time is spent rethinking set-up or initialization parameters. The configuration file is saved on disk under the name "hsmacro.cnf" for Hypersignal-

Macro NRT, Hypersignal-Macro, and Hypersignal-Macro EX versions of Hypersignal software, and as "hsacous.cnf" for Hypersignal-Acoustic software.

The user-environment configuration file includes the current graphics and mouse hardware drivers, current selected acquisition/DSP hardware (real-time software only), current User Setup area, all current menu selections, all current user-modifiable data (filenames, analysis parameters, display options, etc.), current editing cursor position, and so forth. Every user-modifiable parameter in Hypersignal is saved in the configuration file.

The user-environment configuration file can be saved from Hypersignal in three ways:

1) By "exiting" from the main menu ('E' key). Exiting the software saves the configuration file and stops Hypersignal execution.

2) By "updating" from the main menu ('U' key). Updating saves the configuration file but Hypersignal continues to execute.

3) By executing an external program or DOS shell. The configuration file is saved automatically by the program in anticipation of access by the external program.

NOTE: If you "quit" from the Hypersignal main menu, the user-

environment configuration file is not saved.

When Hypersignal next executes, or control is returned to Hypersignal, the user-environment is initialized to a state defined by the configuration file contents. See Hypersignal-Macro Series Users Manual appendix A, User-Environment Configuration File Format, and the "config.doc" file included on the distribution diskettes, for a listing of configuration file contents.

Upon executing Hypersignal software for the first time, a message of the form "cannot find configuration file...Initializing new configuration" should be displayed. This message only appears when executing the software for the first time, because normally the configuration file will be present on disk as a result of performing an "exit" function after a previous Hypersignal session. However, when running Hypersignal for the first time, no configuration file will be present--forcing you to specify graphics and mouse driver selections, since these cannot be determined automatically with 100% accuracy by the software. Please see Hypersignal-Macro Series Users Manual section 2.4, Selecting Graphics and Mouse Drivers, or Graphics and Mouse Driver Selection in Hypersignal User Mode, above, for detailed information on choosing driver selections.

NOTE: Hypersignal checks for a "compatibility identifier" in the configuration save file before reading it. If you have received an updated version of software, there is a good chance your old configuration file won't be "compatible" (due to parameters being changed or added at the end of the file), and the program will display a message of the form "...configuration file not compatible...initializing new configuration..." upon first executing. We apologize for any inconvenience caused by having to reset graphics, mouse, system configuration, and other options.

Below are some notes of interest about accessing the user-environment configuration file:

The example external programs on the distribution diskettes all "include" the "config.doc" file as part of their source code. The "config.doc" file contains an example procedure that searches for the configuration file, opens the file, reads in all parameters, changes a few parameters (by way of example), writes all parameters back to the file, then closes the file.

The current DOS path (drive and subdirectory) is the first place where Hypersignal expects to find the configuration file. Moreover, this is *always* the place where Hypersignal saves its configuration file. If Hypersignal cannot locate a configuration file on the current DOS path, it then searches the path

containing the executable file ("HSxxxxx.EXE") and the DOS path list. If a compatible configuration file is found, then the software will start with this file, but save all new settings in a new file on the current path. Note that this provides the means to set up a multiuser system with multiple configuration files: the initial configuration file can be provided from a master path, and individual, personalized configuration files can be saved to local paths.

Because the current DOS path is used as the default location for the configuration file, this value is not saved inside the configuration file itself. Please do not confuse the current DOS path with the User Setup path (DRIVE and SUBDIRECTORY fields at upper right of main menu). The User Setup path is used to locate all Hypersignal waveforms, and is, of course, saved inside the configuration file. Thus, changing the User Setup path (or the External Program Configuration path or the Macro Manager path or any other path specifiable inside Hypersignal) has no effect on the location of the configuration file.

Hypersignal updates the user-environment configuration file before executing an external program (including DOS shell) and reads the configuration file back again after the external program finishes executing. In this way, a user-created external program can access and modify the

configuration file, causing changes in the Hypersignal environment upon return. In order to guarantee that the configuration file can be read after the external program (or DOS shell) returns, Hypersignal saves the current DOS path (current logged drive and/or subdirectory) before yielding control, and restores the saved value afterwards. If you wish to override the automatic restoration, because external or macro processing needs to change DOS paths, you can set the RESTORE DOS PATH field in the External Program Configuration menu to 'N'. However, keep in mind that if you disable automatic restoration, and then change the current logged drive and/or subdirectory while in an external program or DOS shell, Hypersignal will read (or attempt to read) a new configuration file, if any, after the external program finishes.

For real-time versions of the software, it may be necessary to access the configuration file to change the default memory and/or I/O base address for some types of DSP/Acquisition hardware. Use a text-editor to look at the "config.doc" file (on distribution diskette #2) and find out which parameter in the configuration file to change. Also for real-time versions of the software, there exist a few other parameters affecting DSP/Acquisition board operation that cannot be controlled via data-entry menus, and may need to be accessed in the configuration file.

NOTE: Examples of typical DSP/analog hardware values found in Hypersignal configuration files include the "AIC control word" sent down to boards containing the TLC32044 AIC analog I/O chip at run-time, the "IOC register setting" sent down to DSP32C-based boards at run-time, and codec control words. If you need to change aspects of analog converter, codec, or other analog I/O-related items which can't be controlled from Hypersignal data-entry menus, you should check the configuration file. A value may be available which will let you customize analog I/O operation as needed.

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.9 Creating Waveform Files in Hypersignal

From either user mode or engine mode, there are several ways to create waveform files in Hypersignal:

1) Acquire an existing waveform file in some other format using the File Acquisition function (see Hypersignal-Macro Series Users Manual section 4.1.1, Acquiring Waveforms From Disk Files, or Importing Waveform Files in Hypersignal).

2) Acquire data continuously to disk, in real-time, from a supported DSP/Analog hardware type (see Hypersignal-Macro Series Users Manual section 4.1.2, Acquiring Waveforms From DSP/Acquisition Hardware, or Acquiring Waveform Files From DSP/Analog Hardware in Hypersignal).

3) Use the Difference Equations function to generate a waveform based on input from unit-step, delta, trig, noise, and other functions, combined with recursive input, or "feedback" (see Hypersignal-Macro Series Users Manual section 4.6.2, Function Generation with Difference Equations, or Signal Synthesis with Difference Equations in Hypersignal).

4) Run an external program that, after performing whatever processing for which it was

intended, writes output directly to a waveform file (see Signalogic DSP Software Reference Guide chapter 4, Waveform File Formats).

5) Capture and store to file a display trace from the Hypersignal Waveform Display/Edit function or Digital Oscilloscope or Spectrum Analyzer instrumentation functions (see section Hypersignal-Macro Series Users Manual section 4.7.1, Using Instrumentation Displays).

6) Execute a Hypersignal function, such as Filter Construction, that creates a waveform file as output resulting from its normal operation and processing.

7) Run the tim2wav or wav2tim file conversion macro programs.

Related Topics

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

3.9.1 Creating Complex Waveform Files in Hypersignal

Methods to create complex waveform files in Hypersignal include the following:

1) Acquire an existing complex waveform file in some other format using the File Acquisition function (see Hypersignal-Macro Series Users Manual section 4.1.1, Acquiring Waveforms From Disk Files, or Importing Waveform Files in Hypersignal).

2) Use the "j" operator in the Difference Equations function to generate a complex waveform based on input from unit-step, delta, trig, noise, and other functions, combined with recursive input, or "feedback" (see section 4.6.2, Function Generation with Difference Equations, or Signal Synthesis with Difference Equations in Hypersignal).

3) Run an external program that, after performing whatever processing for which it was intended, writes complex output directly to a waveform file (see Signalogic DSP Software Reference Guide chapter 4, Waveform File Formats).

Related Topics

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.9.2 Creating Multichannel Waveform Files in Hypersignal

Methods to create multichannel waveform files in Hypersignal include the following:

1) Acquire an existing multichannel waveform file in some other format using the File Acquisition function (see Hypersignal-Macro Series Users Manual section 4.1.1, Acquiring Waveforms From Disk Files, or Importing Waveform Files in Hypersignal).

2) Acquire data multichannel continuously to disk, in real-time, from a supported DSP/Analog hardware type (see Hypersignal-Macro Series Users Manual section

4.1.2, Acquiring Waveforms From DSP/Acquisition Hardware, or Acquiring Waveforms From DSP/Analog Hardware in Hypersignal).

3) Run an external program that, after performing whatever processing for which it was intended, writes multichannel output directly to a waveform file (see Signalogic DSP Software Reference Guide chapter 4, Waveform File Formats).

Related Topics

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.9.3 Creating Frequency Domain Waveform Files

Only the FFT Generation function creates frequency domain waveform files directly in Hypersignal. The Difference Equations function could also be used, with careful attention to framesize and organization. If you need to create frequency domain files external to Hypersignal, say for example to specify real and imaginary waveform files upon which an Inverse FFT Generation can then be executed, see Hypersignal-Macro Series Users Manual chapter 5, Hypersignal Waveform File Format, or Signalogic DSP Software Reference Guide section 4.1, Hypersignal Waveform File Format. Although not difficult to generate frequency domain files via external program, you must pay attention to the "frame-based" organization of frequency domain files. One popular example of creating frequency domain files outside of Hypersignal:

Generating complex (".REA" and ".IMA") waveform files that represent the desired magnitude and *phase* response for a filter. An Inverse FFT Generation function can then produce the appropriate time domain impulse response for filtering other waveforms (via Convolution function).

3.10 Finding Waveforms in Hypersignal User Mode

As you begin to build up a large amount of waveform files during the course of your work with Hypersignal, it will become more difficult to locate waveform files. There are three methods you can employ to aide in finding waveform files while using Hypersignal:

1) Popup directory display

2) Utilities menu function "Directory Display"

3) DOS shell

Methods of Locating Waveform Files in Hypersignal

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal with Popup Waveform Directory Display

Popup directory display appears during data-entry editor menus as an automatic window display in the System Status area (upper left of main menu) when a specified filename cannot be found. Also, you can manually invoke popup directory display from inside any data-entry menu by using the 'PgUp' key.

Popup directory display includes waveform filename and extension, length, and date of creation. The search strategy for different waveform extensions and/or more specific waveform filename may be changed via consecutive use of the 'PgUp' key (see Hypersignal-Macro Series Users Manual section 3.4.2, Popup Directory Display, or Popup Directory Display in Hypersignal, above, for complete information on waveform filename extensions and the search strategy).

You may also temporarily enter "*" or "?" wildcard characters in the waveform filename field(s) to further direct a waveform search; for example a 'test*' entry would include all filenames starting with "test", and having the appropriate extension. As another example, an entry of 'x?test' would include all

filenames starting with "x", ending with "test", and having any other character in between.

NOTE: To re-initialize popup directory display after you have changed the filename in the data-entry menu, or added a wildcard character ('*', '?'), *you must reposition the cursor at the beginning of the waveform filename field and 'Enter' the field*--then press the 'PgUp' key.

Related Topics

Finding Waveforms in Hypersignal with Popup Waveform Directory Display

Finding Waveforms in Hypersignal with the Directory Display Menu Iteml

Finding Waveforms in Hypersignal with the Directory Display Menu Item

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

Finding Waveforms in Hypersignal with the Directory Display Menu Item

The Directory Display menu function, listed under the Utilities column heading on the main menu, offers a DOS-like directory search capability. Unless specified explicitly, the drive and subdirectory are assumed to be the current values shown in the User Setup area. Wildcard characters "?" and "*" are allowed; display is shown one page at a time. You can use the PgUp and PgDn keys to see the previous and next pages of directory display, respectively. The last page displayed also includes a "bytes free" display, showing the current number of free space (in bytes) for the current drive. Because of the capability to page backwards, this function is limited to about 100 total pages of display.

If you do not enter a colon (":") in the FILENAME field for the Directory Display menu selection then Hypersignal will assume the DRIVE field from the User Setup area. If you do not enter a backslash ("\") in the FILENAME field then Hypersignal will assume the SUBDIRECTORY field in the User Setup area. Explicitly entering the ':' and/or '\' characters overrides the assumed User Setup values.

For convenient reference, below are listed the basic waveform

filename extensions used by Hypersignal:

Table 5 Partial List of Hypersignal Waveform Filename Extensions

	Time domain waveform data file
	Frequency domain magnitude file
	Frequency domain phase file

Frequency domain power spectra file

Frequency domain complex real file

Frequency domain complex imag file

Related Topics

Finding Waveforms in Hypersignal with Popup Waveform Directory Display

Finding Waveforms in Hypersignal with the Directory Display Menu Item

Finding Waveforms in Hypersignal with the Directory Display Menu Item

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

Finding Waveforms in Hypersignal with the Directory Display Menu Item

Of course, for the most complete file directory service, including all command-line switches offered by DOS in conjunction with the "DIR" command, you can temporarily "shell out" of Hypersignal to the DOS command line ('F10' key)--but you can only do this from the

main menu, and not from inside data-entry menus. Also, when you do this, remember two things:

1) The current DOS path (drive and subdirectory) will most likely be something else than where your waveforms are located. If you change the current DOS path while in the DOS shell, and the RESTORE DOS PATH field in the External Program Configuration menu is disabled, *be sure and change it back.* Otherwise, Hypersignal may read an unwanted configuration file from the new DOS path upon returning. See Hypersignal-Macro Series Users Manual section 3.6, Invoking External Programs and the DOS "Hot-Key", or section 3.7.1, DOS Shell in Hypersignal User Mode, above, for detailed information on the subject of changing the DOS path during DOS shell execution.

2) Type 'Exit' to return to the Hypersignal main menu once you are done in the DOS shell.

Related Topics

Finding Waveforms in Hypersignal with Popup Waveform Directory Display

Finding Waveforms in Hypersignal with the Directory Display Menu Item

Finding Waveforms in Hypersignal with the Directory Display Menu Item

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode
Main Menu in Hypersignal User Mode
Data-Entry Editor in Hypersignal User Mode
Printing Menus and Waveform Displays in Hypersignal User Mode
Running External Programs in Hypersignal
Configuration Files in Hypersignal
Creating Waveform Files in Hypersignal
Finding Waveforms in Hypersignal User Mode
Running Macro Programs in Hypersignal

3.11 Running Macro Programs in Hypersignal

Hypersignal macro language programs can be invoked in three ways:

1) In user or engine mode, from the Macro Manager menu.

2) By invoking Hypersignal from another application, program, or batch file. Hypersignal can be invoked in either user mode or engine mode.

3) By "chaining" to the end of a previous macro program.

Methods of Invoking Hypersignal Macro Programs

General Topics

Command-Line Interface to Hypersignal
Keyboard and Mouse Interface In Hypersignal User Mode
Online Help In Hypersignal User Mode
Main Menu in Hypersignal User Mode
Data-Entry Editor in Hypersignal User Mode
Printing Menus and Waveform Displays in Hypersignal User Mode
Running External Programs in Hypersignal
Configuration Files in Hypersignal
Creating Waveform Files in Hypersignal
Finding Waveforms in Hypersignal User Mode
Running Macro Programs in Hypersignal

3.11.1 Running a Macro Program From the Hypersignal Macro Manager Menu

In user mode, the 'F3' key is used to configure and execute (if ready) a macro from the main menu. When the 'F3' key is pressed, the Macro Manager menu appears, which works like any other Hypersignal data-entry menu, even though it is not listed under one of the three main menu column headings. When the 'Enter' key or a mouse button is pressed while the cursor is on the last field (or the 'ctrl-Enter' key sequence is pressed at any time) when this menu is displayed, the current macro specified in the MACRO FILENAME field is compiled (if needed) and then

executed. Input parameters (if any) used by the macro are also specified in the Macro Manager menu, as is the compiler mode and pathname information needed to find the macro file. Pressing the 'Esc' key from the Macro Manager menu will not execute the currently specified macro. Online Help is available as usual by pressing the 'F1' key.

Depending on the current macro compiler mode specified in the Macro Manager, the macro may or may not be compiled before it actually executes. The macro compiler can be in one of three possible modes: 1) always compile, 2) compile only if necessary ("make mode"), where the macro source code file is only compiled if its date stamp is later than that of the macro object code file, and 3) never compile (use existing macro object code). The software default is the make mode.

Note that the Macro Manager method of running macro programs may also be used if Hypersignal is operating in engine mode. In this case, the external controlling application must send simulated keyboard commands to first invoke the Macro Manager menu, and then set up and execute the desired macro program. DSPower-Block Diagram software uses this technique when re-submitting a macro program (generated for one or more Hypersignal blocks) to a copy of Hypersignal software already active. This method avoids

time spent unloading and loading. Also, if the macro program does not need re-compiling, more time is saved if the built-in Hypersignal compiler is instructed to use the "make mode" described above.

All status (and error messages) concerning macro compilation and execution are displayed in the System Status area (upper left of main menu).

Note that up to 6 (six) input parameters can be specified in the Macro Manager. These input parameters can be used to pass input and output filenames, data-entry menu parameters, and numeric values to a macro, allowing it to be used for different waveforms without modification. Also, the descriptive menu labels can be customized for these parameters for each macro--see the ".MCF" file, below.

For any given macro, Hypersignal keeps track of three separate filename extensions:

1) Source file extension, which defaults to ".MAC", but can be otherwise if specified explicitly in the Macro Manager menu MACRO FILENAME field.

2) Object file extension, which is always ".MXE".

3) Configuration file extension, which is always ".MCF". The configuration file contains all Macro Manager menu entries for the macro in ASCII text form, and is saved whenever the macro is

executed. Since this file is in ASCII form, it can be edited in order to customize the input parameter field labels--see Hypersignal-Macro Series Users Manual section 4.10.1.2, Parameter Passing From the Macro Manager Menu.

See Hypersignal-Macro Series Users Manual chapter 7, HYPERSIGNAL MACRO LANGUAGE, for detailed information on all aspects of macros and the macro language.

NOTE Hypersignal searches for all macro files only on the path (drive and subdirectory) specified in the Macro Manager menu. The User Setup path, or any other Hypersignal path, is not considered.

Related Topics

Running a Macro Program From the Hypersignal Macro Manager Menu

Running a Macro Program From Outside of Hypersignal

Running a Hypersignal Macro Program from Another Macro

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

3.11.2 Running a Macro Program From Outside of Hypersignal

To invoke a macro from outside of Hypersignal, the macro source filename is passed as a command-line parameter in the HSxxxxx.EXE file specification. Hypersignal first ensures that the macro source file exists, and, if so, looks for a macro configuration file (see below) with which to initialize the Macro Manager menu, and then starts the macro. See section 3.1 above, Command-Line Interface to Hypersignal, and section 3.1.2, Command-Line Examples of Running Hypersignal, for more information and examples of command-line parameter specifications. Note that the macro language contains "Abort" and "Exit" commands that allow the program to return from Hypersignal back to the caller, completing a "remote execution" of a macro.

This technique is used by DSPower-Block Diagram software when executing one or more Hypersignal blocks from its block diagram user environment. See

User Mode and Engine Mode in Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components.

Related Topics

Running a Macro Program From the Hypersignal Macro Manager Menu

Running a Macro Program From Outside of Hypersignal

Running a Hypersignal Macro Program from Another Macro

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.11.3 Running a Hypersignal Macro Program from Another Macro

The Macro Manager menu can be invoked from inside an executing macro program by a "Macro Manager" statement, followed by input parameters (if any), interactive commands (if any), and a standard "endfunc" statement (see Hypersignal-Macro Series Users Manual section 7.3, Invoking Hypersignal Menu Functions From Macros). Re-invoking the Macro Manager menu from within an executing macro is an advanced technique that can be used for either of two purposes, or both:

1) Allow the input parameters to the current macro to be changed dynamically. To do this, a $ESC interactive command should be entered prior to the "endfunc" statement, so that the macro does not run "recursively". Changing input parameters dynamically is not recommended as standard macro programming procedure, but could be useful in some situations.

2) To chain another macro to the end of the currently executing macro. To do this, a "Macro Manager" statement, followed by the appropriate new macro filename, any input parameters needed, and an "endfunc" statement, should be entered in the

macro as the *last* executable macro command.

Related Topics

Running a Macro Program From the Hypersignal Macro Manager Menu

Running a Macro Program From Outside of Hypersignal

Running a Hypersignal Macro Program from Another Macro

General Topics

Command-Line Interface to Hypersignal

Keyboard and Mouse Interface In Hypersignal User Mode

Online Help In Hypersignal User Mode

Main Menu in Hypersignal User Mode

Data-Entry Editor in Hypersignal User Mode

Printing Menus and Waveform Displays in Hypersignal User Mode

Running External Programs in Hypersignal

Configuration Files in Hypersignal

Creating Waveform Files in Hypersignal

Finding Waveforms in Hypersignal User Mode

Running Macro Programs in Hypersignal

3.12 Getting Waveforms In and Out of Hypersignal User Mode

This section not incorporated yet. Please see section 4.1 in the Hypersignal-Macro Series Users Manual.

3.13 Displaying Time Domain Waveforms in Hypersignal User Mode

This section not incorporated yet. Please see section 4.2 in the Hypersignal-Macro Series Users Manual.

3.14 Interactive Waveform Displays in Hypersignal User Mode

This section not incorporated yet. Please see section 4.3 in the Hypersignal-Macro Series Users Manual.

3.15 FFT Generation in Hypersignal User Mode

This section not incorporated yet. Please see section 4.4 in the Hypersignal-Macro Series Users Manual.

3.16 Displaying Frequency Domain Waveforms in Hypersignal User Mode

This section not incorporated yet. Please see section 4.5 in the Hypersignal-Macro Series Users Manual.

3.17 Waveform Generation and Mathematics in Hypersignal User Mode

This section not incorporated yet. Please see section 4.6 in the Hypersignal-Macro Series Users Manual.

3.18 DSP/Acquisition Hardware in Hypersignal User Mode

This section not incorporated yet. Please see section 4.7 in the Hypersignal-Macro Series Users Manual.

3.19 Filter Design in Hypersignal User Mode

This section not incorporated yet. Please see section 4.8 in the Hypersignal-Macro Series Users Manual.

3.20 Applying Filters in Hypersignal User Mode

This section not incorporated yet. Please see section 4.9 in the Hypersignal-Macro Series Users Manual.

3.21 Creating Macro Programs and Functions in Hypersignal User Mode

This section not incorporated yet. Please see section 4.10 in the Hypersignal-Macro Series Users Manual.

4 Using DSPower®-HWLib and DSPower-SPLib Software

Overview

[@j Hardware Library, Signal Processing Library]

This section provides information needed to effectively use DSPower-HWLib and DSPower-SPLib software, including:

- background information about the way HWLib and SPLib work
- using the Hardware Manager and Engine Manager DLLs,
- using HWLib and SPLib, including acquiring/generating analog I/O data and calling DSP/math functions
- creating programs which use DSPower-HWLib and DSPower-SPLib DLLs

For a concise description and explanation of individual function calls inside all DSPower-HWLib and DSPower-SPLib DLLs, see

Signalogic DSP Software Reference Guide section 2.2, DLL Functions, to find the function of interest. Also, Function Alphabetic List and Cross-Reference, in the overview section of Signalogic DSP Software Reference Guide chapter 2, Function Reference, contains an alphabetized list of functions.

For more information about media and disk space requirements to use DSPower-HWLib and DSPower-SPLib software, see Signalogic DSP Software System Guide sections 3.1.2, Media and Disk Space Requirements, and 3.1.3, Documentation and Media You Should Have Received.

For information about installing DSPower-HWLib and DSPower-SPLib software, see Signalogic DSP Software System Guide sections 3.2.2, Installing DSPower Series Software, and 3.2.3, Configuration and Checkout of DSPower-HWLib and DSPower-SPLib Software.

Main Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.1 Architecture of DSPower-HWLib and DSPower-SPLib Software

DSPower-HWLib software consists of three (3) DLLs and several support utilities. The DLLs are:

- enmgr.dll (Engine Manager)
- hwmgr.dll (Hardware Manager)
- hwlib.dll (Hardware Library)

DSPower-SPLib software also consists of support files and programs and three DLLs:

- enmgr.dll (Engine Manager)
- hwmgr.dll (Hardware Manager)
- splib.dll (Signal Processing Library)

The Engine Manager and Hardware Manager DLLs are common to both DSPower-HWLib and DSPower-SPLib software. These DLLs are also used by DSPower-Block Diagram software. Essentially, Engine Manager is responsible for controlling and managing DSP Engines transparently to the calling application, and Hardware Manager is responsible for managing the configuration of all DSP/analog hardware installed in the system. For more information, see chapter 7, Using Engine Manager, and chapter 8, Using Hardware Manager.

Related Topics

4.1.1 Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

As with file management and window management in Windows, the concept of handles are used throughout DSPower-HWLib and DSPower-SPLib software. Specifically, the DSEngineOpen and DSAssignBoard calls return unique handles, which implicitly represent the information and parameters used to make these calls. The handles can then be used to simplify the calling convention for many other library calls, and to allow the system to produce profile, status, and error data as needed.

Engine Handles

Engine handles are returned by the DSEngineOpen call. The returned engine handles can then be used in DSAssignBoard calls, which assign

specific hardware types for use by the engine.

Board Handles

Board handles are really "DSP/analog hardware handles", but board handle is an easier term to use in the DSPower-HWLib API. Unique board handles are created by the DSAssignBoard call, and can then be used to access DSP/analog hardware. Examples of DSPower-HWLib calls which require board handles include DSInitBoard, DSResetBoard, DSRunBoard, DSPutMem and DSGetMem, etc.

"DSxxxBoard" calls also have "DSxxxProcessor" equivalents. These calls can used to refer to specific processors on a multiprocessor DSP/analog hardware type. Examples include DSLoadFileProcessor, DSResetProcessor, DSRunProcessor, etc. When used, these calls affect only the processor to which the handle is assigned. The first parameter of these calls, like their DSxxxBoard counterparts, is a board handle. The second parameter of these calls is a "processor number", which is actually a value treated as a series of bits, where each bit refers to a processor (bit 0 == processor 0, bit 1 == processor 1, etc.). This method allows any valid combination of processors to be specified in DSxxxProcessor calls.

Notes

1) DSxxxBoard calls to multiprocessor hardware types affect all processors simultaneously. For example, DSLoadFileBoard loads the same DSP Program File to all processors, and DSRunBoard starts all processors simultaneously (assuming the multiprocessor DSP/analog hardware in question supports this mode).

2) DSxxxProcessor calls will return an error if a processor is specified which is out of range of what is actually on the hardware.

3) The current limit on both engine and board handles is sixteen (16).

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.1.2 Callback Messages in DSPower-HWLib and DSPower-SPLib Software

DSPower-HWLib software allows user-specified window or control handles to be "registered" to several types of DSP Engine callback messages. This is done using the DSRegisterEngineMsgWnd function, which is located in Engine Manager (enmgr.dll). Once registered, window or control handles receive callback messages containing information about, and indicating the status of, ongoing engine and DSP/analog hardware activity.

Below are the current callback message types available in DSPower-HWLib and DSPower-SPLib software:

- engine status messages
- program status messages
- buffer ready messages.
- semaphore messages
- statistics and parameter update messages

A window or control handle must be registered for each of the above callback message types before they can be sent by DSP Engines. Engine and program status messages are handled by Engine Manager (enmgr.dll), and buffer ready, semaphore, and statistics and parameter update messages are

handled by either DSPower-HWLib (hwlib.dll) or DSPower-SPLib (splib.dll).

For detailed information about callback message types, and using the DSRegisterEngineMsgWnd function to enable window or controls to receive these messages, see section 7.4, Registering Callback Messages from DSP Engines.

For source code examples, see the following in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software:

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

Also, the dscope.cpp example source code file included with DSPower-HWLib software implements a digital oscilloscope example which shows the use of DSRegisterEngineMsgWnd and processing subsequent buffer ready messages.

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.1.3 Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

The DSPower-HWLib and DSPower-SPLib error handling scheme is straightforward. Most library calls return a non-zero value on success, and a zero (NULL) on failure. When calls into hwlib.dll (or splib.dll) fail, the DSGetHWLibErrorStatus (or DSGetSPLibErrorStatus) and DSGetEngineErrorStatus calls can be used to attempt to learn more information about the most recent failure. These calls return error and status conditions associated with DSP/analog hardware and DSP/math functions.

Generally, at any given time both of these calls will return the same value. However, there is an

important difference: the latter retrieves error conditions associated with a particular engine, while the former returns an overall DLL value. Thus, if more than one engine is open at one time, it is possible that the DSGetHWLibErrorStatus (or DSGetSPLibErrorStatus) value could be overwritten as a result of consecutive unrelated engine calls.

For error and status conditions associated with engine problems, the DSGetEngMgrErrorStatus and DSGetEngineErrorStatus routines (located in enmgr.dll) should be used. For error and status conditions associated with the Hardware Setup File or Hardware Selector, the DSGetHWMgrErrorStatus routine (located in hwmgr.dll) should be used.

NOTE: The DSGetEngineErrorStatus call is actually inside Engine Manager (enmgr.dll). Note also that Engine Manager maintains an "error/status listbox" window which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of both status and error conditions, including both those resulting from DSPower series DLL calls and generated by DSP Engines. For complete information on error handling in Engine Manager, see section 7.5, Error Handling in Engine Manager.

Below is an example showing error investigation after a DSAssignBoard failure:

```
if (!DSAssignBoard(hEngine, szBoardDesignator, uBusType, uIOBaseAddr,
uMemBaseAddr)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSAssignBoard would be NULL in the case of failure, since a non-NULL value would be a valid board (hardware) handle. Thus, to find out more information, the DSGetEngineErrorStatus, DSGetEngMgrErrorStatus, and DSGetHWLibErrorStatus calls are checked. Note that DSGetEngineErrorStatus requires a valid engine handle as an input parameter. Thus, for problems related to a DSEngineOpen call, the DSGetEngineErrorStatus function should be avoided.

Below is a partial list of error values which can be either returned by DSGetEngineErrorStatus or DSGetHWLibErrorStatus, along with a brief description of their meaning. These constants can be found mostly in the hwlib.h file and partly in the dllerr.h file. For a list of Engine Manager and Hardware Manager error values, see sections 7.5.1, DLL Error Handling in Engine Manager, and 8.4.1, DLL Error Handling in Hardware Manager. For a complete list of all DSPower series software error

conditions by error number, see Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions.

dllerr.h or hwlib.h constant	Summary of Meaning
DSDSPFILENOTFOUND	

D
S
H
A
R
D

WARENOTRESPONDING

DSINVALIDBOARDHANDLE

D
S
C
O
U
L
D
N
O
T
O
P
E
N
T
E
M
P
L
A
T
E
F
I
L
E

DSCOULDNOTCREATESOU

R
C
E
F
I
L
E

DSCREATINGE

NGINEPROGRAMERROR

DSUNABLETOALLOCATEMEMO

R
Y

D
S
U
N
A
B
L
E
T
O
R
E
A
D
T
E
M
P
L

A
T
E
F
I
L
E

DSERRORINDATAPARAM

D
S
B
O
A
R
D
D
O
E
S
N
O
T
S
U
P
P
O
R
T
C
A
L
L

DSINVALIDWORDLENGTH

DSINVALIDM

E
M
O
R
Y
T
Y
P
E

DSINVALIDBOARDDESIGNATOR

DSBOARDNOTINHWSETUPFIL

E

DSALLBOARDHANDLESALLOCATED

DSINVALIDPROCESSORNUMBER

DSCOULDNOTINITIALIZEDLL

Note that in C/C++ programs, it is safest to treat the above error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions, for more information about this grouping.

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.1.4 Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

Like any software package, there are several items and tradeoffs to consider when attempting to optimize the performance of DSPower-HWLib and DSPower-SPLib software. Below are some areas to keep in mind.

Performance Considerations Topics

Continuous Disk Transfer Using DSPower-HWLib

Changing DSP Programs Using DSPower-HWLib

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Continuous Disk Transfer Using DSPower-HWLib

When implementing continuous transfer to display or disk using DSPower-HWLib software, the following are points to consider:

1) Video and display considerations mentioned in Signalogic DSP Software System Guide section 1.4.3, Video and Display Considerations, are applicable. In addition, because Windows 3.x graphics is not known for its speed, the number of points and window size makes a big difference. For example, plotting a full-scale waveform on a maximized window will take much longer to draw than if the waveform amplitude is small or the window size is reduced. Furthermore, if there are more x-axis points than the window is wide, you may want to consider an algorithm that reduces the actual number of x-axis points by averaging or taking the maximum amplitude of each group of points that map to the same display pixel.

2) When transferring data continuously between disk and DSP/analog hardware, bigger is better for buffer sizes. Especially with caching utilities such as SMARTDRV.EXE, or with 32-bit disk access enabled, there are many possibilities for disk drive delays. Sometimes it is not at first obvious

why these "disk drive performance enhancers" can have detrimental effects on long-period continuous data transfer; for some interesting observations, see DSP/Analog Hardware Performance Considerations, in Signalogic DSP Software System Guide section 1.4.2, Performance Considerations. In general, it is true that bigger buffers help absorb disk drive delays. Note that when using a Hypersignal-Macro series software as a DSP Engine, the maximum buffer size possible is currently 32768.

3) DSP/analog hardware data transfer speed is critical. There are a wide variety of methods of data transfer implemented on hardware types that are in common use, with an equally wide variety of transfer speeds. Essentially, DSP/analog hardware which offers some sort of inherent "host port" method of data transfer provide the fastest rates; additionally, these methods usually operate reliably regardless of whether the DSP device is running. Hardware with fully dual-ported SRAM is also a fast option, but can entail some unpleasant system problems if a memory-mapped addressing method is used. For a review of DSP/analog hardware data-transfer methods, see chapter 2, Hardware Considerations, in the Hypersignal-Macro Series Hardware Reference Guide.

Performance Considerations Topics

Continuous Disk Transfer Using DSPower-HWLib

Changing DSP Programs Using DSPower-HWLib

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

Changing DSP Programs Using DSPower-HWLib

The fastest way to start a new DSP function is to already have it loaded in DSP board memory, and simply do the following:

change the reset vector value in the DSP code, using DSPutMem

reset the DSP/analog hardware, using DSResetBoard or DSResetProcessor

Another alternative is to download new DSP code to an area of DSP program memory currently not being used, and again change the reset vector and reset the board. Note that this method is only possible if the DSP/analog hardware supports program memory access from the host PC while the DSP is running. In the worst case, the DSP must be stopped, new code downloaded, and the DSP restarted.

Performance Considerations Topics

Continuous Disk Transfer Using DSPower-HWLib

Changing DSP Programs Using DSPower-HWLib

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.2 Creating Programs for DSPower-HWLib and DSPower-SPLib

The best advice that can be given for creating your first programs which use DSPower-HWLib or DSPower-SPLib software is to look at the source code and project file examples provided with the software. That being said, this section and its related topics discuss the creation of programs using DSPower-HWLib and DSPower-SPLib software in detail.

Project files are included with DSPower-HWLib and DSPower-SPLib software for Borland C/C++ v3.1 and v4.5, Microsoft Visual C/C++ v1.5, and WATCOM C/C++[32] v10.0. These project files and compiler versions have been tested at Signalogic. You should be able to load these project files directly into your IDE, and compile them immediately. From there, it should be straightforward to either create modified versions of these projects, or create new projects that incorporate the necessary elements.

Note that the demo programs all invoke the Hardware Selector, which allows user-selection of the DSP/analog hardware type on which to run the demos. The Hardware Selector is provided mainly for convenience, and does not necessarily need be used in user-defined applications. The

return value from the DSShowHardwareSelectorDlg call is a Board Designator string, which can then be used in DSAssignBoard calls to generate a board handle. Alternatively, DSAssignBoard calls can easily be "hardcoded" for a specific hardware type.

The following basic elements might typically be found in a Windows application using DSPower-HWLib or DSPower-SPLib software. The steps marked with ‡ can be performed in either the WinMain section of the application, or in a WM_CREATE message of a window procedure:

- ‡ optional: call DSShowHardwareSelectorDlg to obtain a user-specified Board Designator
- ‡ open a DSP Engine using the DSEngineOpen call
- ‡ assign a board handle with DSAssignBoard using the engine handle returned by DSEngineOpen and the Board Designator returned by DSShowHardwareSelectorDlg; alternatively, "hardcode" the Board Designator string directly, and...
  - ‡ perform a DSInitBoard call

to verify the presence of the selected DSP/analog hardware and ensure it is responding at the correct I/O and/or memory base address(es)

‡ perform a DSResetBoard call

‡ perform a DSLoadFileBoard call to download a desired DSP Program file

create a window to receive "buffer ready" messages from the DSP Engine

respond to buffer ready callback messages to the window procedure and process data coming from or going to the DSP/analog hardware

or...

use the engine handle from DSEngineOpen in high-level functions such as

DSAcquireWvfrmFile or DSGenerateWvfrmFile

For source code examples showing actual implementation of some of the above items, see section 4.2.4 below, Example Code: DSPower-HWLib and DSPower-SPLib Software.

Related Topics

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.2.1 Hardware Library and Signal Processing Library Access

[@k implicit loading, explicit loading]

To access Hardware Library and/or Signal Processing Library, the hwlib.dll and/or splib.dll libraries must be visible to your application's calling program. The easiest way to do this is to include the hwlib.lib and/or splib.lib files in your project. For C/C++ applications, please see the

example project files, including Borland C/C++ v3.1 and v4.5, Microsoft Visual C/C++ v1.5, and WATCOM C/C++32 v10.0, provided with DSPower-HWLib software. For MATLAB projects, please see the example .m files. Once included in your project file, and thus linked into your final executable, the .lib files cause their corresponding DLLs to be loaded and initialized when your executable first runs (actually, *before* any code in the executable runs). This is known as implicit loading.

Another way of accessing Hardware Library or Signal Processing Library is to load them from your application program dynamically, using the LoadLibrary call in the Windows API, followed by one or more GetProcAddress calls with specific function names as input parameters. This method is known as explicit loading, and is more complicated, requiring painstaking typecasting to create function prototype *pointers* which are set as a result of GetProcAddress calls. "Behind the scenes", this is also the technique used by MATLAB to load DLLs needed by calling .m programs--except that MATLAB simplifies the process by allowing DLL calls to look much like other MATLAB function calls.

Also, for C/C++ programs, the hwlib.h and/or splib.h include files must be specified along with other include files that may be needed by

the calling application. Note that both the hwlib.h and splib.h include files expects the standard Windows 3.1 windows.h include file to have been previously specified.

For more information about using library and include files, see sections 4.2.2 and 4.2.3 below, Include Files for DSPower-HWLib and DSPower-SPLib, and .LIB Files for DSPower-HWLib and DSPower-SPLib.

Related Topics

Hardware Library and Signal Processing Library Access

Include Files for DSPower-HWLib and DSPower-SPLib

.LIB Files for DSPower-HWLib and DSPower-SPLib

Example Code: DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.2.2 Include Files for DSPower-HWLib and DSPower-SPLib

[@k include files]

For C/C++ programs using DSPower-HWLib software, the first two, and possibly the third, of

the following include files must be specified:

- hwlib.h or splib.h (Hardware Library or Signal Processing Library)
- enmgr.h (Engine Manager)
- hwmgr.h (Hardware Manager)

In some cases, hwmgr.h may not be necessary. This would occur if the calling application made no provision for operators to change or store parameters regarding DSP/analog hardware configuration, and thus made no calls to the Hardware Setup File.

The above include files needed by the calling application must be specified along with other Windows or standard C./C++ include files that may be required. Note that all of the above include files expect the standard Windows 3.1 include file windows.h to have been previously specified.

NOTE: For C/C++ development, make sure the include file directory list in your IDE contains the DSPower-HWLib subdirectory. The example project files supplied with DSPower-HWLib software assume default include file subdirectories of "hwlib", "toollib", and "helplib" under the root-level installation subdirectory, although toollib and helplib are present only for demo purposes.

Related Topics

Hardware Library and Signal Processing Library Access

Include Files for DSPower-HWLib and DSPower-SPLib
.LIB Files for DSPower-HWLib and DSPower-SPLib
Example Code: DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software
Creating Programs for DSPower-HWLib and DSPower-SPLib
DSPower-SPLib Function Overview
Acquiring Data Using DSPower-HWLib Software
Using Engine Manager
Using Hardware Manager

4.2.3 .LIB Files for DSPower-HWLib and DSPower-SPLib

[@k library files]

Example project files are included with DSPower-HWLib and DSPower-SPLib software for Borland C/C++ v3.1 and v4.5, Microsoft C/C++ v1.5, and WATCOM C/C++[32] v10.0. These project files include references to the following library files:

- hwlib.lib or splib.lib (Hardware Library or Signal Processing Library)
- enmgr.lib (Engine Manager)
- hwmgr.lib (Hardware Manager)

Either the hwlib.lib or splib.lib library files, and the enmgr.lib and hwmgr.lib files, must be specified in the project files of calling applications in order to allow

compile-time linking and cause run-time implicit loading of the enmgr, hwmgr, and hwlib and/or splib DLLs. Even if no calls are made to the Hardware Manager DLL by the application, the hwmgr.lib file should still be specified in the project file, since the hwlib, splib, or enmgr DLLs may contain such calls.

Implicit loading under Windows means that any DLLs used by an application are automatically loaded and initialized when the application runs (this occurs before any code in the application itself executes).

NOTE: Enmgr.lib, hwmgr.lib, hwlib.lib and/or splib.lib library files are absolutely required in your project file if creating a new application. References to DSPower-HWLib or DSPower-SPLib calls will produce link-time errors if these library files are not "visible" in the application's project file. The example project files supplied with DSPower-HWLib software assume default library file subdirectories of "hwlib", "toollib", and "helplib" under the root-level installation subdirectory, although toollib and helplib are present only for demo purposes.

Related Topics

Hardware Library and Signal Processing Library Access

Include Files for DSPower-HWLib and DSPower-SPLib

.LIB Files for DSPower-HWLib and DSPower-SPLib

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.2.4 Example Code: DSPower-HWLib and DSPower-SPLib Software

Below are several source code examples illustrating fundamental concepts in DSPower-HWLib and DSPower-SPLib software. These are presented here mainly for convenience, especially so for the online Help version of documentation, which allows an opportunity to jump back and forth between the examples and individual function descriptions in Signalogic DSP Software Reference Guide section 2.2, DLL Functions. Also, Function Alphabetic List and Cross-Reference, in the overview section of Signalogic DSP Software Reference Guide chapter 2, Function Reference, contains an alphabetized list of functions. For the most complete source code examples, please refer directly to

the .cpp files included with DSPower-HWLib and DSPower-SPLib software.

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Related Topics

Hardware Library and Signal Processing Library Access

Include Files for DSPower-HWLib and DSPower-SPLib

.LIB Files for DSPower-HWLib and DSPower-SPLib

Example Code: DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

Example C/C++ Code: Include Statements for DSPower-HWLib

Below is an excerpt from the dscope.cpp source code example file included with DSPower-HWLib software. It shows a typical include file section which allows compilation of applications using the DSPower-HWLib software. Remember that referring to the prototypes inside the DSPower-HWLib DLL include files shown below will cause the

necessary .lib (library) files to be needed in the application's project file at link-time; for more information on DSPower-HWLib and DSPower-SPLib .lib files, see section 4.2.3 above, .LIB Files for DSPower-HWLib and DSPower-SPLib.

```
// typical application-specific include files
#include <windows.h>
#include <stdlib.h>
#include <math.h>

// DSPower-HWLib DLL include files
#include "enmgr.h"   // Engine Manager
#include "hwmgr.h"   // Hardware Manager
#include "hwlib.h"   // Hardware Library

// more Signalogic DLL include files, but used only for user-interface in demo
#include "toollib.h"
#include "helplib.h"
#include "minmax.h"  // local versions of min/max, if needed
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Initializing DSP/Analog Hardware

Below is a source code example similar to the dscope.cpp file, showing some engine and DSP/analog hardware initialization typical of working with DSPower-HWLib.

```
char szBoardDesignator[50];
HENGINE hEngine;
HBOARD hBoard;
DWORD dwFsMode, dwMemSize;
WORD wBoardClass;
UINT uMemArch;
float FsActual;

// exit WinMain if nothing to do in demo
if (DSShowHardwareSelectorDlg(NULL, szBoardDesignator) == IDCANCEL)
goto cleanup;

// open engine
hEngine = DSEngineOpen(DS_EO_HSM, NULL);

// first try to open Hypersignal-Macro or Hypersignal-Macro EX as engine
if (!hEngine) {

    // if that doesn't work, try Hypersignal-Acoustic
    hEngine = DSEngineOpen(DS_EO_HSA, NULL);

    if (!hEngine) { // still an error?

        short int err = DSGetHWLibErrorStatus();
        itoa(err, tmpstr, 10);
        MessageBox(NULL, tmpstr, "Dscope Demo:  DSEngineOpen failed",
        MB_OK);
        goto cleanup;
    }
}

// assign board handle:  engine handle, bd designator, bus type, IO/mem base
addrs
hBoard = DSAssignBoard(hEngine, szBoardDesignator, NULL, NULL, NULL);

// initialize the board; make sure it's installed and responded
if (!DSInitBoard(hBoard)) {
```

```
    itoa(DSGetEngineErrorStatus(hEngine), tmpstr, 10);
    MessageBox(NULL, tmpstr, "Dscope Demo:  DSInitBoard failed", MB_OK);
    goto cleanup;
}

// interrogate engine for board type values
wBoardClass = DSGetBoardClass(hBoard);

// get memory architecture
uMemArch = DSGetMemArch(hBoard);
if (uMemArch == NULL) {

    MessageBox(NULL," ","Dscope Demo: DSGetMemArch failed", MB_OK);
    goto cleanup;
}

// find samp rate ctrl. reg. value and actual rate (closest rate possible to desired)
// 22.05 kHz sampling rate, assume simultaneous sampling

dwFsMode = DSCalcSampFreq(hBoard, 22050.0, NULL, NULL, &FsActual);

// load DSP Program File (usually a COFF file produced by DSP chip linker)
if (!DSLoadFileProcessor(hBoard, NULL, 0x01)) {  // load default file (proc 0 only)

    MessageBox(NULL, " ", "Dscope Demo:  problem loading file", MB_OK);
    goto cleanup;
}

// get the memory size, (note that this currently has to be done after LoadFile)
dwMemSize = DSGetMemSize(hBoard, 0x01);

// reset the DSP board (processor 0 only)
DSResetProcessor(hBoard, 0x01);

// send down some important variables (needed if a default Hypersignal DSP file
is being used)
...
...
DSPutHVarMem(hBoard, DSP_BOARDCLASS, wBoardClass & 0x00ff);
DSPutHVarMem(hBoard, DSP_BOARDSUBCLASS, wBoardClass >> 8);
...
DSPutHVarMem(hBoard, DSP_FSMODE, dwFsMode);
...

...initialize more DSP code variables, flags, buffers, etc. if necessary..

// scope window is born; turn the board loose...
DSRunProcessor(hBoard, 0x01);  // processor 0 only
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Creating a Callback Window

Below is another source code example taken from dscope.cpp, showing partial code for creating a window and designating it to receive buffer ready messages:

```
// create window that can be used for callback messages
hwnd = CreateWindow(wndclass.lpszClassName,
        "Example DSPower-HWLib Test Program",
        WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL |
        WS_CLIPCHILDREN | WS_VISIBLE,
        rectScopePos.left,
        rectScopePos.top,
        nWidth,
        nHeight,
        NULL,
        NULL, // no menu
        hInstance,
        NULL);

ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);

// scope window is born; turn the board loose...
```

```
DSRunProcessor(hBoard, 0x01);  // processor 0 only

// register window to receive buffer ready messages, tell engine to wait for input
data

DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForBuffer(hBoard, 0, NULL, DS_WFB_POLLED);

// no choice now but to follow Alice...
while (GetMessage(&msg, NULL, 0, 0)) {

    TranslateMessage (&msg);
    DispatchMessage (&msg);
};
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Processing Buffer Ready Messages

Below is a source code example from dscope.cpp showing some typical processing upon receiving a buffer ready message:

```
WORD wBuflen;
DWORD dwBufferBaseAddr;
int nCurBuf;
UINT uStatus;
```

```
int far* pBuf;

case WM_DSPENGINE_BUFRDY:  // buffer ready message sent by DSP engine

    pBuf = (int far*)GlobalLock(hBuf);  // lock memory to be used

    // transfer below illustrates two common types of DSP memory architectures

    if (DSGetMemArch(hBoard) == DS_GMA_VECTOR) {  // vector data memory

        if (nCurBuf == 0)  uStatus = DSGetMem(hBoard,
        DS_GM_VECTOR_DATA_X, dwBufferBaseAddr, DS_GM_SIZE16,
        pBuf, wBuflen);

        else  uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_Y,
        dwBufferBaseAddr, DS_GM_SIZE16, pBuf, wBuflen);
    }
    else { // linear data memory

        uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
        dwBufferBaseAddr+nCurBuf*wBuflen, DS_GM_SIZE16, pBuf, wBuflen);
    }

    GlobalUnlock(hBuf);  // let memory float

    if (!uStatus) MessageBox(NULL, "DSGetMem:  problem with point transfer",
    "Dscope Test Prog", MB_OK);

    nCurBuf ^= 1;  // switch buffers
    DSPutHVarMem(hBoard, DSP_HOSTBUFNUM, nCurBuf);// write new buffer
    #

    // may we have another? please...
    DSWaitForBuffer(hBoard, nCurBuf, NULL, DS_WFB_POLLED);
    break;
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Below is a partial source code example from dtape.cpp showing registration of a callback window or control handle to receive update messages:

```
// register window to receive various messages from engine
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETENGINESTATUSMSG |
DS_REMW_SETSTATISTICSMSG | DS_REMW_SETBUFNUMMSG |
DS_REMW_SETLEVELMSG, hdlg);
```

Note that only one callback window or control handle can be assigned to an engine at one time. Thus, another DSRegisterEngineMsgWnd call using a new handle would essentially redirect future update messages to the new handle.

Below are partial source code examples similar to the dtape.cpp example file included with DSPower-HWLib software showing some typical processing of update messages:

```
case WM_DSPENGINE_PROGSTATUS:

    // if engine has entered idle state due to error, finishing the instructed
    function,
    // etc, process STOP button to reflect the current status

    if (State != IDLE && wParam == DS_PS_PROGDONE) {

        SendMessage(hdlg, WM_COMMAND, DS_ATC_TAPESTOP, 0L);
```

```
    }
    break;

case WM_DSPENGINE_SAMPFREQ:

    // get IEEE floating-point actual sampling rate from engine message

    float Fs;
    _fmemcpy(&Fs, &lParam, sizeof(float));  // engine sends IEEE single-float
    value

    char tmpstr[30];
    gcvt(Fs, 10, tmpstr);  // convert f.p. value to string

    UpdateStatusBar(hSamplingRate, tmpstr, "Actual Sampling Rate");
    break;

case WM_DSPENGINE_BUFSIZE:

    dwBufferSize = lParam;  // store the buffer length being used
    break;

case WM_DSPENGINE_BUFNUM:

    dwCurSamplePos = BufferSize*lParam;  // calculate current sample count

    char tmpstr[30];
    ltoa(dwCurSamplePos, tmpstr, 10);  // convert to text

    UpdateStatusBar(hSampleCounter, tmpstr, "Current Sample Count");
    break;

case WM_DSPENGINE_LEVEL:

    UpdateMeter(hMeter, SETMETERLEVEL, lParam);  // lParam has amplitude
    break;
```

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Below is a partial source code example showing unregistration of window or control handles currently receiving specified types of update messages:

```
// unregister windows receiving various messages from engine
```

```
DSUnregisterEngineMsgWnd(hEngine, DS_REMW_SETENGINESTATUSMSG |
DS_REMW_SETSTATISTICSMSG | DS_REMW_SETBUFNUMMSG |
DS_REMW_SETLEVELMSG);
```

Note that a window or control handle is not needed in the DSUnregisterEngineMsgWnd call because only one callback window handle for each callback message category can be actively registered at any one time. In the example above, whatever number of window handles were registered using DSRegisterEngineMsgWnd calls are unregistered (but only for the callback message categories specified).

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Acquiring Waveform Files

Below is a partial source code example from dtape.cpp showing typical setup and usage for the DSAcquireWvfrmFile call. The

dtape.cpp demo attempts to open Hypersignal-Macro or Hypersignal-Acoustic software as a DSP Engine, but the source code example below applies regardless of the engine. One exception may occur in the szFilename parameter of the CONVERSIONINFO structure; for more information, see section 4.5 below, Acquiring Data Using DSPower-HWLib Software.

```
// create CONVERSIONINFO object memory; use zero-initialization to set most
parameters to default values

hConvInfo = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE |
LMEM_DISCARDABLE, sizeof(CONVERSIONINFO));

pConvInfo = (PCONVERSIONINFO)LocalLock(hConvInfo);

lstrcpy(pConvInfo->szFilename, szWaveformFilename);  // file to acquire

// setup for one channel, init channel list (example is sequential)

pConvInfo->NumChan = 1;          // one channel
for (j=0; j<pConvInfo->NumChan; j++) pConvInfo->ChanList[j] = j;

// setup up sampling rate, amount, gain, trigger information

pConvInfo->Fs =22050;  // 22.05 kHz sampling rate
pConvInfo->NumSamples = 100000L;  // 100,000 samples
lstrcpy(pConvInfo->szGainList, "1");  // see notes in CONVERSIONINFO
pConvInfo->NumTrigChan = 1;  // one trigger channel
pConvInfo->TrigChanList[0] = 0;  // freerun trigger

// start acquisition

DSAcquireWvfrmFile(hBoard, pConvInfo);

// free and delete memory

LocalUnlock(hConvInfo);
LocalFree(hConvInfo);
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Below is a partial source code example from dtape.cpp showing typical setup and usage for the DSGenerateWvfrmFile call. The dtape.cpp demo attempts to open Hypersignal-Macro or Hypersignal-Acoustic software as a DSP Engine, but the source code example below applies regardless of the engine. One exception may occur in the szFilename parameter of the CONVERSIONINFO structure; for more information, see section 4.5 below, Acquiring Data Using DSPower-HWLib Software.

```
// create CONVERSIONINFO object memory; use zero-initialization to set most parameters to default values

hConvInfo = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE | LMEM_DISCARDABLE, sizeof(CONVERSIONINFO));

pConvInfo = (PCONVERSIONINFO)LocalLock(hConvInfo);

lstrcpy(pConvInfo->szFilename, szWaveformFilename);  // file to play

// NOTE:  FOR D/A OUTPUT ONLY, a 0 value for number of channels, sampling frequency, and number of samples causes engine to "auto configure", initializing these values from waveform file header
```

```
pConvInfo->NumChan = 0;
pConvInfo->Fs = 0;
pConvInfo->NumSamples = 0;

lstrcpy(pConvInfo->szGainList, "1");  // see notes in CONVERSIONINFO

// start playback

DSGenerateWvfrmFile(hBoard, pConvInfo);

// free and delete memory

LocalUnlock(hConvInfo);
LocalFree(hConvInfo);
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

4.3 DSPower-HWLib Function Overview

The subsections listed below in this section provide a brief overview of the main groups of DSPower-HWLib functions. For detailed information about each function, Signalogic DSP Software

Reference Guide section 2.2, DLL Functions, is the place to look. Also, Function Alphabetic List and Cross-Reference, in the overview section of chapter 2, Function Reference, contains an alphabetized list of all functions. If you are using Signalogic DSP Software online Help, then perform the following steps:

select Program Manager | Reference Guide (in the Signalogic DSP Software group)

select "Search", and enter the first few characters of the function you are looking for

if you are in another online Help file besides Reference Guide, find and click on a topic starting with "Reference..." to bring up a quick reference section to Reference Guide topics

Another place to look for quick reference information on DSPower-HWLib functions are the following include files:

- enmgr.h (Engine Manager DLL)
- hwmgr.h (Hardware Manager DLL)
- hwlib.h (DSPower-HWLib DLL)

Keep in mind also that before any of the functions listed below in this section are used, DSEngineOpen or

DSAssignBoard calls (or both) must have been previously invoked. The engine and board handles returned by these calls are needed by other functions in DSPower-HWLib. For additional information, section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software.

Main DSPower-HWLib Function Groups

4.3.1 Hardware and Processor Control

Board and processor control functions are the lowest-level function available in the DSPower-HWLib DLL. Some of these functions essentially amount to "touching" DSP/analog hardware directly. Below is a partial list of these functions.

Board and Processor Control Functions

Initialize: DSInitBoard

Reset: DSResetBoard and DSResetProcessor

Run: DSRunBoard and DSRunProcessor

Run: DSRunBoard and DSRunProcessor

Hold State: DSHoldBoard and DSHoldProcessor

Shutdown: DSDisableBoard

Related Topics

Hardware and Processor Control
DSP Program File Downloading
Hardware Interrogation
DSP Memory Data Transfers
Executing High-Level Functions

Initialize: DSInitBoard

The DSInitBoard call first checks for the presence of the specified DSP/Analog hardware. If found, it resets all accessible registers, and places the processor into a known state. Usually this simply means the DSP processor is held in reset; however, for some hardware, a short "host interface" segment of code is downloaded and the processor is left running. In these instances, the short code segment usually involves onchip DMA transfer from host PC "communication registers", on an interrupt-driven basis. These cases occur for DSP/analog hardware types which do not support direct access to onboard DSP memory; example hardware types include DSP5600x, and a few TMS320C3x, 'C4x, and 'C5x.

For multiprocessor hardware, all processors are initialized identically (and simultaneously, if the hardware supports this mode).

Board and Processor Control Functions

Initialize: DSInitBoard
Reset: DSResetBoard and DSResetProcessor

Run: DSRunBoard and DSRunProcessor

Run: DSRunBoard and DSRunProcessor

Hold State: DSHoldBoard and DSHoldProcessor

Shutdown: DSDisableBoard

Related Topics

Example C/C++ Code: Initializing DSP/Analog Hardware

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Reset: DSResetBoard and DSResetProcessor

The DSResetBoard call places the DSP device in reset. Other registers and resources are not affected.

For multiprocessor hardware, DSResetProcessor resets all processors (simultaneously, if the hardware supports this mode).

The DSResetProcessor version is intended for multiprocessor hardware and has an additional parameter which allows any combination of processors on the DSP/analog hardware to be placed in reset.

Board and Processor Control Functions

Initialize: DSInitBoard

Reset: DSResetBoard and DSResetProcessor

Run: DSRunBoard and DSRunProcessor

Run: DSRunBoard and DSRunProcessor

Hold State: DSHoldBoard and DSHoldProcessor

Shutdown: DSDisableBoard

Related Topics

Example C/C++ Code: Initializing DSP/Analog Hardware

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Run: DSRunBoard and DSRunProcessor

The DSRunBoard call removes the DSP device from a reset state, allowing it to run. Typically, this means the device will execute its start-up reset sequence, including jumping the code location specified in its reset vector.

For multiprocessor hardware, DSRunBoard allows all processors to run (simultaneously, if the hardware supports this mode).

The DSRunProcessor version is intended for multiprocessor hardware and has an additional parameter which allows any combination of processors on the DSP/analog hardware to be allowed to run.

Board and Processor Control Functions

Example C/C++ Code: Initializing DSP/Analog Hardware

Initialize: DSInitBoard

Reset: DSResetBoard and DSResetProcessor

Run: DSRunBoard and DSRunProcessor

Run: DSRunBoard and DSRunProcessor

Hold State: DSHoldBoard and DSHoldProcessor

Shutdown: DSDisableBoard

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Hold State: DSHoldBoard and DSHoldProcessor

The DSHoldBoard call places the DSP device in hold state, if it supports this mode. Currently, hold state is applicable only for some DSP/analog hardware with TMS320Cxx devices, and even some of these cases, hold state is not supported by the board design.

For multiprocessor hardware, DSHoldBoard places all processors in hold state (simultaneously, if the hardware supports this mode).

The DSHoldProcessor version is intended for multiprocessor hardware and has an additional parameter which allows any combination of processors on the

DSP/analog hardware to be placed in hold state.

Board and Processor Control Functions

Initialize: DSInitBoard

Reset: DSResetBoard and DSResetProcessor

Run: DSRunBoard and DSRunProcessor

Run: DSRunBoard and DSRunProcessor

Hold State: DSHoldBoard and DSHoldProcessor

Shutdown: DSDisableBoard

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Shutdown: DSDisableBoard

The DSDisableBoard call places the DSP device in reset, and disables any additional resources on the board that may be used by the DSP code driver. Examples could include disabling resources accessible from the host PC, such sampling rate generation timers, or issuing a "shutdown" interrupt to the DSP code which causes it to disable resources not accessible from the PC.

For multiprocessor hardware, DSDisableBoard resets all

processors (simultaneously, if the hardware supports this mode).

Board and Processor Control Functions

Initialize: DSInitBoard

Reset: DSResetBoard and DSResetProcessor

Run: DSRunBoard and DSRunProcessor

Run: DSRunBoard and DSRunProcessor

Hold State: DSHoldBoard and DSHoldProcessor

Shutdown: DSDisableBoard

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

4.3.2 DSP Program File Downloading

The DSLoadFileBoard call downloads a specified executable DSP file to the all processors on the DSP/analog hardware.

The DSLoadFileProcessor version is intended for multiprocessor hardware and has an additional parameter which allows any combination of processors to be specified to receive the executable DSP file. Note that in some cases, even if not specified in the call, it may be unavoidable downloading

executable files to DSP devices on a multiprocessor board because they may share program memory. Such a case does not necessarily mean that processors are running the same program, only that programs being run by another processor may be "visible" somewhere in program memory.

Related Topics

Example C/C++ Code: Initializing DSP/Analog Hardware

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

4.3.3 Hardware Interrogation

Board interrogation functions allow DSP/analog hardware drivers to report back important information about the hardware. This can involve some sophisticated detection on the part of the drivers, such as determining how much memory has been installed on the DSP board. Below is a partial list of these functions.

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Hardware Classification: DSGetBoardClass

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Memory Size: DSGetMemSize

The DSGetMemSize function returns the amount of data memory detected on the specified DSP processor on the specified DSP/analog hardware. For most hardware types, the return value is in units of the "natural wordlength" of the DSP device. Some examples include 32-bit words for TMS320C3x and TMS320C4x devices, 24-bit words for DSP5600x devices, and 16-bit words for ADSP-21xx and TMS320C5x devices. However, there are currently two (2) exceptions. The DSGetMemSize return value is in 16-bit words for DSP32C and DSP3210 devices, even though these devices are 32-bit floating-point. This is because these devices have true multiwidth access to onchip and external memory, and some DSP Engine drivers take advantage of this feature to store data (especially

analog I/O data) in compact form when possible.

Notes

In cases where the DSP/analog hardware has separate program and data memory areas, the DSGetMemSize function does not measure program memory.

In some cases, the return value may either be somewhat less than or exceed the amount of installed onboard, external data memory. This may be because external memory addressing is contiguous with onchip memory, or overlaps onchip memory (creating an unusable region).

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Hardware Classification: DSGetBoardClass

Related Topics

Hardware and Processor Control

Memory Architecture: DSGetMemArch

The DSGetMemArch function returns the type of memory architecture used by the DSP device on the DSP/analog hardware specified by the input board handle. Following are the possible return values:

linear, both program and data in the same memory space

modified Harvard architecture, with program and data linear in separa

te memory spaces

modified Harvard architecture with program and data in separate memory spaces, and with data memory further subdivided into vector memory spaces

Notes

This function does not actually "touch" hardware. The return value is completely determined by what the information in the Hardware Setup File

entry for the specified DSP/analog hardware type.

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Hardware Classification: DSGetBoardClass

Related Topics

Example C/C++ Code: Processing Buffer Ready Messages

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Word Length: DSGetWordLength

[@j natural wordlength]

The DSGetWordLength function returns the natural word length used for data memory by the DSP device on the DSP/analog hardware specified by the input board handle. Following are the possible return values:

indicates 16-bit fixed-point

data memory

indicates 24-bit fixed-point data memory

indicates 32-bit floating-point data memory

Notes

This function does not actually "touch" hardware. The return value is completely determined by what the information in the

Hardware Setup File entry for the specified DSP/analog hardware type.

This function is convenient for use with DSGetMem or DSPutMem functions when transferring integer data between DSP data memory and host PC memory.

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Hardware Classification: DSGetBoardClass

Related Topics

Example C/C++ Code: Processing Buffer Ready Messages

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Sampling Frequency Calculation: DSCalcSampFreq

Given a desired sampling frequency, the DSCalcSampFreq

function return two important pieces of information:

- A mode or control register value that can be sent to resident DSP code and used to program the DSP/analog hardware sampling rate clock generators. High-level functions in DSPower-HWLib use this value implicitly.
- The actual sampling rate that the DSP/analog hardware will use, given the desired sampling frequency as input. This value is important because often times only a limited range of discrete sampling rates are possible.

Below is an example of using the DSCalcSampFreq call:

```
DWORD FsMode;
float ActualFs;
FsMode = DSCalcSampFreq(hBoard, 44100, NULL, NULL, &ActualFs);
```

The example above asks for a sampling rate of 44.1 kHz, and assumes simultaneous sampling, which requires no entry for number of channels and channel list (see notes below). The closest sampling frequency value to 44.1 kHz that can be achieved by the hardware is returned in ActualFs, and the corresponding value needed to program the hardware's clock generator is returned in FsMode.

Notes

The actual sampling rate is returned in a float value pointed to by a function call parameter, while the mode value is the function return value.

The number of channels and channel list input parameters are significant only for DSP/analog hardware which does not utilize simultaneous sampling for all channels. Examples include "channel scan" and multiplexing channel groups into single A/D converters. If you know the specified DSP/analog hardware has simultaneous sampling in all cases, the number of channels parameter and channel list parameter can be left NULL.

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Hardware Classification: DSGetBoardClass

The DSGetBoardClass function returns a hardware classification value suitable for use by DSP Source Code Interface software. This value is maintained by DSP Engines because DSP Source Code Interface software for one type of DSP chip family may support multiple types of DSP/analog hardware, each with different configuration and analog I/O driver characteristics. It can be sent to executing DSP source code during or prior to run-time initialization. See Signalogic DSP Software Reference Guide section 3.2, Board Designator, Driver ID, and Classification Table.

The result of the DSGetBoardClass function can be sent directly to the Hypersignal DSP Source Code software DSP_BOARDCLASS and DSP_BOARDSUBCLASS variables. See Signalogic DSP Software Reference Guide section 3.2, Board Designator, Driver ID, and Classification Table.

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Hardware Classification: DSGetBoardClass

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

4.3.4 DSP Memory Data Transfers

Data transfer functions allow C/C++ and MATLAB programs to transfer data between local or global host PC memory and DSP onchip or external (onboard) memory. The workhorse data transfer functions in DSPower-HWLib are the DSPutMem and DSGetMem calls. These functions allow one or more data elements of variable length to be transferred to/from various segments of DSP memory. All types of available DSP memory are accessible, with the possible exception of DSP chip onchip memory, which may only

accessible from the DSP device under certain circumstances.

Also available are the DSPutHVarMem and DSGetHVarMem calls, which allow pre-defined variables and semaphores used in Hypersignal DSP Source Code Interfaces to be transferred to and from DSP memory. For more information, see section 5.0.2, DSP Source Code Interfaces for Hypersignal.

Data Transfer Functions

Memory Block Transfers: DSPutMem and DSGetMem

Hypersignal DSP Code Variable Transfers: DSPutHVarMem and DSGetHVarMem

Related Topics

Example C/C++ Code: Processing Buffer Ready Messages

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Memory Block Transfers: DSPutMem and DSGetMem

The DSPutMem and DSGetMem calls allow data elements to be transferred between host PC memory and DSP/analog hardware memory. One or more data elements can be transferred to or from a specified address in DSP external memory, or in onchip DSP

memory for some types of hardware. Depending on the hardware, it may be possible to control the width of the data elements. Below are some examples:

```
if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

    uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_X, Addr,
    DS_GM_SIZE16, pBuf, wBuflen);
}
else { // linear data memory

    uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT, Addr,
    DS_GM_SIZE16, pBuf, wBuflen);
}
```

The above examples show reading from DSP data memory at a location specified by Addr. wBuflen data elements of 16-bits each are read. The first example assumes a hardware type with vector data memory, and the second linear data memory (see Memory Architecture: DSGetMemArch in section 4.3.3 above, Hardware Interrogation). The return value should be checked for success.

Data Transfer Functions

Memory Block Transfers: DSPutMem and DSGetMem

Hypersignal DSP Code Variable Transfers: DSPutHVarMem and DSGetHVarMem

Related Topics

Example C/C++ Code: Processing Buffer Ready Messages

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

Hypersignal DSP Code Variable Transfers: DSPutHVarMem and DSGetHVarMem

The DSPutHVarMem and DSGetHVarMem functions allow variables and semaphores to be transferred between C/C++ and MATLAB programs and pre-defined locations maintained by Hypersignal DSP Source Code Interfaces for various DSP devices. Hypersignal DSP Source Code Interfaces include several real-time operating modes, as well as a number of math and DSP routines. For detailed information, see section 5.0.2, DSP Source Code Interfaces for Hypersignal.

Below are some examples:

```
DSPutHVarMem(hBoard, DSP_BOARDCLASS, wBoardClass & 0x0ff);
DSPutHVarMem(hBoard, DSP_OPMODE, 2);  // Dig. Scope is mode 2
DSPutHVarMem(hBoard, DSP_BUFLEN, wBuflen);  // buffer size
DSPutHVarMem(hBoard, DSP_NUMCHAN, 1);     // number of channels
DSPutHVarMem(hBoard, DSP_FSMODE, FsMode);  // sampling rate mode
value
```

The above examples are taken from the dscope.cpp example file; in the this file, hBoard is previously created by a DSAssignBoard call to a DSP/analog hardware type chosen from the Hardware Selector. The first example writes the board classification returned by DSGetBoardClass, to indicate to the Hypersignal DSP source code what hardware is being used.

Succeeding examples set up the operating mode, buffer length, number of channels, and sampling rate clock generation value (see DSCalcSampFreq). These examples are only small portion of what is needed to completely enable one of the built-in operating modes in the Hypersignal DSP Source Code Interfaces.

Data Transfer Functions

Memory Block Transfers: DSPutMem and DSGetMem

Hypersignal DSP Code Variable Transfers: DSPutHVarMem and DSGetHVarMem

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

4.3.5 Executing High-Level Functions

High-level functions in DSPower-HWLib are defined as functions which are handled completely by the DSP Engine, given a minimal set of input parameters, with no monitoring or updating required by the calling application. For example, an engine might be instructed to acquire a waveform file of a specified length with specified trigger criteria.

Currently, there are two types of high-level engine functions available in DSPower-HWLib software:

- data acquisition and measurement functions
- general DSP Engine functions

Data acquisition and measurement functions are implemented by first initializing a CONVERSIONINFO structure, and then passing a pointer to this structure as a parameter to the DSAcquireWvfrmFile or DSGenerateWvfrmFile functions. These flexible functions allow simple operations such as acquiring and playing back waveform files, as well as more complicated operations such as stimulus & response measurement and real-time digital filtering. See section 4.5 below, Acquiring Data Using DSPower-HWLib Software.

General DSP Engine functions can be executed using the DSExecEngineFunction found in Engine Manager. Engine functions can be executed synchronously or asynchronously, meaning the calling application either waits for the engine function to complete, or continues executing simultaneously. The DSExecEngineFunction allows any DSP Engine function to be executed, using a command string--essentially as if the engine was being operated in user mode. Making DSP Engine functions available in this way offers great flexibility, but not great ease-of-

use. Unlike the data acquisition and measurement functions available from DSPower-HWLib, to successfully use DSExecEngineFunction, you need to know the details of the engine function you are trying to invoke. For more information, see section 7.3, Executing Arbitrary Functions in DSP Engines.

Related Topics

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

4.4 DSPower-SPLib Function Overview

This section not available yet.

4.5 Acquiring Data Using DSPower-HWLib Software

Analog data can be acquired (or output) to/from supported DSP/analog hardware using DSPower-HWLib software in the following ways:

Using the DSAcquireWvfrmFile and DSGenerateWvfrmFile calls in hwlib.dll. For examples, see Example C/C++ Code: Playing Back Waveform Files, and Example C/C++ Code: Acquiring Waveform Files, in section 4.2.4 above, Example Code: DSPower-HWLib and DSPower-SPLib Software.

Using the DSGetMem or DSPutMem calls in hwlib.dll, assuming that an appropriate DSP Program File has been downloaded to the DSP/analog hardware beforehand.

Executing a DSP Engine function that acquires or generates analog I/O data to/from waveform file or DSP board memory in some other way.

Related Topics

CONVERSIONINFO Structure

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

CONVERSIONINFO Structure

[@j CONVERSIONINFO structure, CONVERSIONINFO, PCONVERSIONINFO]

Below is a reconstruction of the CONVERSIONINFO structure, which is used in high-level DSPower-HWLib calls such as DSAcquireWvfrmFile and DSGenerateWvfrmFile. The reprint below is partially divided into the following sections:

- basic acquisition/generation information
- real-time digital filter information
- trigger information
- stimulus & response (simultaneous input/output information)

Each element in the structure is commented with a brief description. For more detailed information, see the additional explanations of some elements after the structure.

```
typedef struct {
// basic acquisition/generation information
```

/ / real-time digital filter information

```
// trigger information
```

```
// stimulus & response information
```

```
} CONVERSIONINFO;

typedef CONVERSIONINFO FAR*
PCONVERSIONINFO;
```

Notes

1) The entire CONVERSION structure should be initialized to zero before using, so that default values can take effect for parameters not used. This can easily be accomplished using a call such as

```
_fmemset(&ConversionInfo, 0, sizeof(CONVERSIONINFO));
```

2) Previous value used in either DSEngineOpen or DSSetEngineWaveformPath call is used for waveform filenames, unless explicit path information is found in the filename above.

NOTE: For operation when a Hypersignal-Macro series program is the DSP Engine, szFilename above should not contain path information. The path should instead be set using the initial DSEngineOpen call, or a subsequent DSSetEngineWaveformPath call.

---

3) The szGainList parameter is a string because input (or output) gain and/or attenuation can be specified as a software digital scaling value, or as a hardware gain value in Volts or dB, depending on the DSP/analog hardware type.

Also, entry can be channel-specific, or specified as a channel-range.

4) TrigDelay parameter not yet operational.

5) PCONVERSIONINFO can be used to declare pointers to CONVERSIONINFO structures. For examples, see Example C/C++ Code: Playing Back Waveform Files, and Example C/C++ Code: Acquiring Waveform Files, in section 4.2.4 above, Example Code: DSPower-HWLib and DSPower-SPLib Software.

Related Topics

CONVERSIONINFO Structure

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

5 Using DSP Source Code Interface Software

Overview

DSP Source Code Interface software can be categorized into four general areas:

1. DSP and math functions
2. ISRs (analog I/O drivers) for particular DSP/analog hardware
3. operating mode infrastructure code to support instruments, interactive displays, and certain types of simulation or acceleration of host processing
4. real-time operating system and related utilities

There are two versions of Signalogic DSP Source Code Interface software: source code necessary to provide Hypersignal-Macro series software functionality (to support DSP/analog hardware related functions, displays, and instruments), and source code necessary to provide the same type, as well as code generation functionality, for other Signalogic

DSP Software components. For a description of Hypersignal DSP Source Code Interfaces, see section 5.0.2, DSP Source Code Interfaces for Hypersignal. For a revision history list, see Signalogic DSP Software System Guide section 3.3.4, Revision History for DSP Source Code Interface Software.

NOTE: A signed Non-Disclosure Agreement (NDA) is required prior to obtaining DSP Source Code Interface software for a given DSP chip. Basically, the NDA requires work with the DSP Source Code to be for in-house R&D and product-development purposes, and prohibits using the DSP Source Code within, or to help develop, a product competing with Signalogic DSP Software components. If you would like to see a copy of the NDA, please call and ask for one to be faxed to you.

In order to make effective use of DSP Source Code Interface software, appropriate assemblers/linkers are required. C/C++ compilers are optional, but highly recommended if you plan to modify/add a substantial amount of code (especially for floating-point devices). However, keep in mind that compiler output--even that of the best optimizing compilers--is always at least 3-5x slower than hand-optimized code, which can make the difference between a system operating in real-time or not. Taking compiled code and optimizing only time-critical

sections can prove an effective compromise.

In all cases, DSP Source Code Interface software has been implemented and tested with the *DSP device manufacturer's development tools* (i.e. assembler, linker, etc.). The version of tools used for testing is indicated in the individual DSP device family sections listed below. Development tools from other vendors may be compatible, but Signalogic assumes no responsibility if not.

Related Topics

Specific DSP Source Code Interface Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

AT&T WE DSP32 Source Code Interface for Hypersignal

AT&T WE DSP32C Source Code Interface for Hypersignal

AT&T WE DSP3210 Source Code Interface for Hypersignal

Motorola DSP5600x Source Code Interface for Hypersignal

Motorola DSP96002 Source Code Interface for Hypersignal

NEC 77220 Source Code Interface for Hypersignal

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

5.0.1 DSP Source Code Interface Documentation

Many parts of the sections concerning Hypersignal DSP Source Code Interfaces in this chapteronline Help file are more or less reprinted from chapter 4 of the Hypersignal-Macro Series Hardware Reference Guide, but with the addition of information helpful in placing DSP Source Code Interface software in perspective with other Signalogic DSP Software components. Additional information includes noting differences the different versions of DSP Source Code Interfaces, when Hypersignal functionality can affect other components using Hypersignal as a DSP Engine (see User Mode and Engine Mode, in Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components), and precise locations of subjects in all forms of Signalogic DSP Software documentation.

NOTE: Sections reprinted from Hypersignal-Macro Series Hardware Reference Guide chapter 4 may have material that copyrighted © 1985-1990 by Hyperception, Inc., and/or copyrighted © 1991-1995 by Signalogic, Inc. All rights are reserved by their respective copyright holders.

Also, these sections provide a convenient source of online Help when working with DSP Source Code Interface software (editing, running a software simulator, etc.) under Windows. To quickly access this help when operating DOS or protected-mode DOS development tools (e.g. DSP chip manufacturer development tools), do the following:

1. run the development tool from its Windows icon, or from a DOS prompt started from Program Manager window

   once the development tool is running, use the Alt-Tab keyboard sequence to switch back to Progam Manager, leaving the development tool software active

   select this online Help section, and make it full screen

   now use Alt-Tab to switch back to the development tool; use Alt-Tab again to re-activate the online Help when needed

---

NOTE: DSP Source Code Interface documentation assumes a knowledge of DSP chip programming and mastery of development tool software, including assembler, linker, compiler, software simulator, downloaders, and debuggers.

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

Hypersignal DSP Source Code License Agreement

DSP Source Code Interfaces for Hypersignal in Engine Mode

5.0.2 DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface software for Hypersignal includes both library routines and also real-time, interrupt-driven structures that allow foreground and background processing in conjunction with sample interrupts. Initially, the source code supports standard Hypersignal operation, including instrumentation displays, continuous transfer to/from disk, and file-based acceleration, with no programming required. In other words, if you were to assemble and link the source code for a specific DSP chip as it was shipped, without any changes, you would end up with the distribution object file that is sent with every basic Hypersignal package. However, the source code could then be modified and/or added to in order

to enhance the baseline Hypersignal operations or add new functions entirely. Changes can be as simple as interfacing to new analog converters or as complicated as simulating and testing complete algorithms by transferring data between the DSP board and Hypersignal software resources (including hard disk and video display) using pre-defined buffer-switching mechanisms and real-time interrupt structures.

NOTE: The DSP Source Code Interface distribution diskette for a given DSP chip family is not normally included in Hypersignal software shipments, unless purchased additionally or separately.

Related Topics

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

Hypersignal DSP Source Code License Agreement

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP Source Code Interface Application

Examples Using Hypersignal

A basic example of using DSP Source Code Interface software with Hypersignal is to perform some type of custom algorithm or feature extraction. At any time, Hypersignal can be instructed to use a specified executable DSP file(s) as needed, causing user-written algorithm(s) to execute. This type of modification process basically allows you to concentrate on the problem at hand, since the Hypersignal high-level host software performs housekeeping associated with controlling the DSP board, downloading executable code, and transferring blocks of real-data between host and DSP board memory. Furthermore, if you maintain compatibility with internal Hypersignal data and waveform file formats, then more housekeeping tasks such as file creation, graphical analysis, etc. can be left to Hypersignal.

Following are some more detailed examples of using the DSP Source Code Interface with Hypersignal. Lets suppose that you had one of the DSP boards supported by Hypersignal, but needed to interface the board to a target analog section of your own design. If you had the DSP source code for the appropriate chip family, this would be trivial. First, you would change the interrupt service routine for the Hypersignal function that you wanted to get working. Depending on the chip family there

is generally a few different interrupt service routines for each function, organized by board manufacturer. The changes you would make here would be minor: ensuring that the code read your A/D port address instead of the supported board's standard address. Next, you might have to change the system initialization module if the interrupt source/level for your analog hardware is different. Everything else would be left alone: buffering of the data, transfer to the high-level host software, and processing of the data (although you might want to change that, too, at some point). Next, you would assemble the modules changed using the "ASM" batch file provided for each chip family. Finally, you would invoke the "LNK" batch file, and copy the resulting linker output file (".OUT", ".BIN", ".HEX", etc. depending on manufacturer convention) to your working Hypersignal directory. When you next invoked Hypersignal, this file would be ready for downloading and subsequent execution without any further effort.

As another example, suppose you wanted to take a supported board, and working with the board's standard analog I/O, implement all or part of a custom algorithm on the board and stream the real-time results to hard disk for later evaluation and display. By instructing Hypersignal to process 2-channel data, and applying your algorithm to every other sample in

the input buffers as they become ready, you can achieve convenient real-time storage to disk, and comparison viewing and evaluation in both time and frequency domain of input data vs. algorithm results.

As a third example, consider using a supported board to simulate an entire project, maintaining just enough of the Hypersignal interface and real-time data transfer mechanism to allow real-time display of data, debug modes, and possible streaming of data to disk. After transferring code to your target system (via EPROM), there can be no doubt as to whether encountered problems are due to the software or the target hardware design--the Hypersignal simulation can always be easily run at any time as a check against the software design.

Or, consider a case where the DSP Source Code Interface is used as the basis for software in a new product or system, with embedded intermediate data "probe" points and real-time transfer mechanisms left installed. By connecting a computer to the product in some way (RS-232 or parallel connection), a laptop or notebook computer can with Hypersignal installed could be brought into the field and used for testing, maintenance and/or data collection at any time.

Other examples include adding DSP source code to support multiprocessing, interconnections between multiple boards, and other

peripheral bus standards such as DT-Connect, measuring algorithms or parts of algorithms against a particular real-time sampling rate, evaluating the effects of finite word-length, and more.

Working with DSP Source Code Interface software for Hypersignal is very straightforward. The fact that there exists similar implementations for nine (9) different DSP chip families has led to standardization of library calls and operational procedures, from the function level down to the batch files used to facilitate the code modification process. For example, all DSP chip families have similar routines available for post-FFT processing: log, squareroot, arctan, etc. And, at a lower level, all "LNK.BAT" files produce a map file of the form "XXXLINK.MAP" where XXX is "T30" for Texas Instruments TMS320C3x, "M56" for Motorola DSP5600x, "N20" for NEC 77220, etc. Moreover, the modification and download process is also standardized between DSP families. For example, Hypersignal parses and downloads output files from each DSP manufacturer's linker to the DSP board for subsequent execution, obviating the need for user-written object file downloaders.

Most interrupt service routines fully save and restore machine context to protect anything they might modify, and most routines (such as the "snap-in" real-time filter routines that we have

included in the source code library) also save and restore anything they might destroy. In all chip families the Digital Oscilloscope and Spectrum Analyzer functions provide the best examples of full context-save interrupt service routines combined with substantial foreground processing constructed from many of the routines in the source code library.

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

Hypersignal DSP Source Code License Agreement

DSP Source Code Interfaces for Hypersignal in Engine Mode

DSP Source Code Access And Control From Hypersignal

Hypersignal-Macro series software packages contain capabilities which facilitate access and control of Hypersignal DSP source code containing user-defined modifications or additions, including:

1) The ability to specify different executable DSP files to load and run when a DSP-based function or display is invoked from Hypersignal. In the System

Configuration menu, entering a filename, such as "new.out", after the board designator will override the Hypersignal default filename.

2) A macro language function (WriteDSPVal) that allows a macro to cause Hypersignal menu and internal values to be written to arbitrary locations in DSP memory.

3) A macro language function (WriteDSPMem) that allows a macro to cause arbitrary values to be written to arbitrary locations in DSP memory.

For more information about the latter two functions, see Hypersignal-Macro Series Users Manual section 7.10, Macro Commands and Functions.

Note that both the WriteDSPMem and WriteDSPVal functions always make full-word accesses, depending on the natural word-length of the DSP device. This can affect the address value used, if the DSP is capable of accessing data pieces smaller than full-word length. Also, note that both functions are capable of accessing all memory types supported by the DSP chip. For a complete list of internal Hypersignal values that can be written to arbitrary DSP source code memory, see Signalogic DSP Software Reference Guide section 3.3, Hypersignal DSP Source Code Variable List, and for fully detailed information about the syntax and usage of these functions, see Hypersignal-Macro Series Users

Manual section 7.10, Macro Commands and Functions.

NOTE: For the most flexibility in manipulating and accessing executing DSP source code, the DSPower-HWLib DLL should be used. See section 5.0.3 below, DSP Source Code Interfaces for Hypersignal in Engine Mode

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

Hypersignal DSP Source Code License Agreement

DSP Source Code Interfaces for Hypersignal in Engine Mode

Hypersignal DSP Source Code Notes

[@k trace, channel]

Below are some miscellaneous notes to keep in mind when working with Hypersignal DSP Source Code Interface software.

1) There are several references to "trace" and "channel" in DSP source code and source code variable descriptions. Traces are references to waveforms in displays or waveform files, where channels refer to the physical analog input or output connection on the

DSP/analog hardware. A logical-to-physical mapping takes place when channel lists are sent from Hypersignal software to executing DSP code.

Hypersignal DSP Source Code License Agreement

Compliance with the following license agreement is one of the provisions in the NDA agreement mentioned above, a signed copy of which is required before delivery of a DSP Source Code Interface can take place:

Hyperception, Inc. and Signalogic, Inc. reserve all rights to source code supplied with "Hypersignal" series software.

USE OF THIS SOURCE CODE BY YOU ("YOU" MEANS THE COMPANY AND/OR INDIVIDUALS THAT ARE USING THE "HYPERSIGNAL" SOFTWARE) FOR PURPOSES OF RESALE OR PROFIT, OR FOR INCLUSION IN, EITHER WHOLE OR IN PART, ANY OTHER SOFTWARE INTENDED FOR SALE OR PROFIT, IS PROHIBITED.

You are free to copy source code sections to make backup copies and to perform experimentation and testing during algorithm and product development and research. If it is discovered that you, through resale, through lending, or allowing through carelessness or inaction, have enabled another party to obtain possession of a section of the DSP source code marked with a Hyperception or Signalogic copyright, which the other party is including within, or with, a product competitive with the "Hypersignal" series software, then HYPERCEPTION AND SIGNALOGIC ARE PREPARED TO PROSECUTE TO THE FULLEST EXTENT OF THE LAW.

YOU ACKNOWLEDGE THAT YOU HAVE READ THE ABOVE AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT AND THE HYPERSIGNAL DSP SOURCE CODE NON-DISCLOSURE AGREEMENT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF ANY AGREEMENT BETWEEN US, INCLUDING ANY OTHER ORAL OR WRITTEN COMMUNICATION.

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

Hypersignal DSP Source Code License Agreement

DSP Source Code Interfaces for Hypersignal in Engine Mode

5.0.3 DSP Source Code Interfaces for Hypersignal in Engine Mode

When a Hypersignal-Macro series software package is being used as a DSP Engine by another Signalogic DSP Software component, no inherent type of host PC communication or fixed variable/flag locations are assumed. A Hypersignal DSP Source Code Interface executable file is an option, but is certainly not required.

If Hypersignal DSP Source Code *is* used, for example if a built-in DSP source code operating mode such as continuous disk acquisition or digital oscilloscope is desired, then the information in section 5.0.2 above, DSP Source Code Interfaces for Hypersignal, still applies.

However, access to the executable DSP code while it is running is no longer dependent on a built-in Hypersignal instrument or function. For example, DSPower-HWLib offers DSPutMem and DSGetMem calls which can be used to send data back and forth to pre-defined buffers or variable locations in DSP onboard or onchip memory. There are also DSPutHVarMem and DSGetHVarMem calls which can be used to access pre-defined Hypersignal variables, if needed. See chapter 3, Using DSPower-HWLib and DSPower-SPLib Software, for more information.

Furthermore, executable DSP files produced from any DSP source code, whether based on a Hypersignal DSP Source Code Interface or not, can be downloaded and run.

General DSP Source Code Interface Topics

DSP Source Code Interface Documentation

DSP Source Code Interfaces for Hypersignal

DSP Source Code Interface Application Examples Using Hypersignal

DSP Source Code Access And Control From Hypersignal

Hypersignal DSP Source Code Notes

5.1 Texas Instruments TMS320C25 Source Code Interface

Main Topics:

Related Topics:

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface Modification Notes

5.1.1 Texas Instruments TMS320C25 Source Code Interface for Hypersignal

The Hypersignal TMS320C25 DSP Source Code Interface distribution diskette contains all TMS320C25 source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic TMS320C25 source code includes analog I/O interrupt service routines for the LSI/Spectrum, Ariel DSP-16 and PC-C25 (formerly called the DSP-C25), Texas Instruments SWDS, and Burr-Brown PCI-20202C TMS320C25 boards.

As is, the TMS320C25 source code is written in modular form, and supports the basic "modes" of operation below:

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the TMS320C25 boards and generally match up with the foreground processing modules above. Interrupt service routine module names will have "ISR" in the middle of the filename, after two letters indicating the hardware they support (ISRs are usually hardware-specific), and before a digit indicating the operating mode (see section 5.1.1 above, Texas Instruments TMS320C25 Source Code Interface for Hypersignal) that they support, for example "LSISR0". Different interrupt service routines use different interrupt levels for different boards. Examine the interrupt vector and mask initialization in the "T25DEF" (non-executable definitions) and the "T25BEG" module, depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

TMS320C25 Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she must keep in mind the necessary links between Hypersignal software and the TMS320C25 code. The two-way communication between the PC and TMS320C25 established in the Hypersignal software must not be changed, as the high-level host software expects to read/write certain memory locations. These memory locations are initialized by table reads from program memory inside module T25BEG, and are suffixed by 'ADD', as in MODADD, TRGADD, etc. *Do not insert any code of any type before these addresses that would cause them to move*. The TMS320C25 source code has been documented to aid the user in these areas.

For all TMS320C25 boards, you should avoid writing to program memory between 20H and 3FFH, which is typically left open for TMS320C25 debuggers, and 42DH and 4FFH, which is used by Hypersignal to store host-DSP communication variables and structures. Also, note that if Hypersignal DSP TMS320C25 code is allowed to run, it stores

local variables in onchip data memory locations 0-127, location 2FF, and 896 through 1023 (filter coefficients and delay elements).

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

TMS320C25 Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5" for one example of a real-time, interrupt-driven structure.

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

5.1.2 TMS320C25 Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal TMS320C25 DSP Source Code Interface distribution diskette:

variable & constant definition

beginning vectors, including ISR vectors, and including initialization, including PC-written

variables

continuous A/D ISR for LSI/Spectrum board

continuous A/D ISR for Ariel DSP-16 board

continuous A/D ISR for Texas Instruments SWDS board

continuous A/D ISR for Ariel PC-C25 (formerly DSP-C25) board (with TLC3204x AIC)

continuous
A/D ISRs for
Burr-Brown
PCI-20202C

continuous
D/A ISR for
LSI/Spectrum
board

continuous D/A ISR for Ariel DSP-16 board

continuous D/A ISR for Texas Instruments SWDS board

continuous D/A ISR for Ariel PC-C25 (formerly DSP-C25) board (with TLC3204x AIC)

continuous D/A ISR for Burr-Brown PCI-20202C

Digital Scope ISR for all TMS320C25 boards.

Spectrum Analyzer ISR for all TMS320C25 boards.

Stimulus & Response ISR for all TMS320C25 boards.

foreground A/D-to-disk processing (mode=0)

foreground A/D-to-disk processing (mode=0; Burr-Brown board only)

foreground D/A-from-disk processing (mode=1)

foreground Digital Oscilloscope processing (mode=2)

FFT file-based acceleration processing (mode=3)

iFFT file-based acceleration processing (mode=4)

foreground Spectrum Analyzer process (mode=5)

foreground stimulus & response (mode=6)

convolution/correlation file-based acceleration processing (mode=7)

arctangent routine

convolution/correlation routine

fractional and integer divide routines

exponentiation (base-2) routine (needed for fixed-point transfer function calculation)

complex FFT/iFFT routine of variable length N (N=2**m)

post-FFT processing (mag, phase, scaling)

invokes correct filter for current channel	
FIR filter routine	
IIR filter cascade struc 2 (modified direct-form)	
IIR filter cascade struc 3 (balanced pole-zero)	
log2 approximation routine	

magnitude routine

find maximum routine

find minimum routine

move data

4-quadrant phase routine (–π to +π; –pi to +pi, complex input)

sine value lookup-table for FFT use

post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation

squareroot routine

apply pre-defined window (used with FFTs)

complex-ratio	routine used for transfer function

utility. Each .ASM module above must be assembled first.

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

5.1.3 TMS320C25 Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of TMS32025.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to TMS320C25 source code, keeping in mind the necessary shared variables between Hypersignal and the TMS320C25. The file T25DEF.ASM is well-documented in this area.

2) Assemble all modules changed and any new modules created (using Texas Instruments "COFF" assembler) to ensure that a new

.OBJ (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that can be invoked as:

```
ASM filename <car ret>
```

to assemble source code modules (entering the ".ASM" extension after "filename" is optional). This file executes a

```
dspa -l -v25 %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the T25LINK.CMD control file (use an ASCII text-editor to update the T25LINK.CMD file).

3) Re-link all modules using Texas Instruments TMS320C25 "dsplnk" linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ files. LNK.BAT executes a

```
dsplnk T25LINK.CMD
```

statement, where "T25LINK.CMD" is an ASCII linker-command file that specifies all .OBJ files to be linked by the dsplnk.exe program, as well as specific information about memory addresses for some sections. Note that all executable code that is re-locatable starts at 400H. All code has been tested using version 5.05 of "dspa.exe" and "dsplnk.exe".

4) Copy the resulting "TMS32025.OUT" file to directory with Hypersignal software, run Hypersignal, and invoke the appropriate functions. Hypersignal reads the "COFF" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information

TMS320C25 Source Code Interface Modification Notes

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

TMS320C25 Source Code Interface Modification Notes

1) If you need to change default buffer addresses, please consult the comments and suggestions in the T25DEF.ASM file.

2) The LNK.BAT file produces a "T25LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module

included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

3) The T25LINK.CMD file creates an uninitialized "hole" from 20H through 3FFH that can be used for monitor and debugger programs. Because this section is uninitialized, the Hypersignal COFF loader will not write anything to this area.

4) If you need to use an external sampling clock, follow the basic guidelines below:

Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.

Enter '-EXT' after the board designator in the Hypersignal System

Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". The value is in

Hz (Hertz); note that for TMS320C25 boards, which use 16-bit wide memory accesses, the maximum value that can be represented is limited to 65535. This value is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

Texas Instruments TMS320C25 Source Code Interface for Hypersignal

TMS320C25 Source Code Interface Interrupt Service Routines

TMS320C25 Source Code Interface Host PC Communication

TMS320C25 Source Code Interface Real-Time Processing Structure

TMS320C25 Source Code Interface File List for Hypersignal

TMS320C25 Source Code Modification Process for Hypersignal

5.2 AT&T WE DSP32 Source Code Interface

Main Topics:

Related Topics:

AT&T WE DSP32 Source Code Interface for Hypersignal

DSP32 Source Code Interface Modification Notes

5.2.1 AT&T WE DSP32 Source Code Interface for Hypersignal

The DSP32 DSP Source Code Interface distribution diskette contains all AT&T WE DSP32 source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic DSP32 source code includes hardware support code for Communications Automation and Control (CAC) and Burr-Brown ZPB32 WE DSP32 boards.

As is, the DSP32 source code is written in modular form, and supports the basic "modes" of operation below:

Related Topics

AT&T WE DSP32 Source Code Interface for Hypersignal

DSP32 Source Code Interface File List for Hypersignal

5.2.2 DSP32 Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal DSP32 DSP Source Code Interface distribution disk:

	contains sections for all functions
	generates new DSP32.BIN file from DSP3SIM simulator ASCII output file
	generates new DSP32.BIN file from DSP32 board program memory (assumes board is DS3EMU emulator-compatible)

Related Topics

5.2.3 DSP32 Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of DSP32.BIN before running the "LNK" batch file in case your new code doesn't work.

If you wish to alter the performance or function of the code, you must keep in mind the necessary links between the high-level Hypersignal host software and WE DSP32 code. The two-way communication between the PC and DSP32 established in the high-level host Hypersignal software must not be changed, as the high-level software expects to read/write certain memory locations. The WE DSP32 source code has been documented to aid the user in this area.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to DSP32 source code, keeping in mind the necessary links between Hypersignal and the DSP32. All steps below refer to source code as filename.s. If you start with the source code included on DSP Source Code Interface distribution,

then filename.s = DSP32.S in the steps below.

The user may find it convenient to create a batch file to execute the following steps 2-6.

2) Re-assemble the source code using the AT&T "make utility" (assembler and linker) by typing:

```
DSP3MAKE -m2 filename.s
<car ret>
```

Add the switch "-l" after "-m2" if you desire a listing file output. Add the switch "-s" if simulator testing will be required--but you will need to move the current FFT window data space at on-chip RAM F000H (memory mode 2) because the simulator wants to store breakpoint code at FE00H. The "asm.bat" file does the same thing as the DSP3MAKE command listed above; type:

```
ASM filename <car ret>
```

3) Run the AT&T simulator program to create a file of op-codes. Type:

```
DSP3SIM filename.o <car
ret>
```

where filename is the name of the source program.

After the "$im:" prompt, type

```
mem[0:last_byte].b>"DSP
32.BYT" <car ret>
```

where "last_byte" is the address of the last byte of program code, or enter a value, for example 0x1000, that ensures all program code is written to disk (The default amount

for the source code in its initial form is 0x1000. If you add a significant amount of code, or change the "buf0adr" base address value, you may need to change the amount of simulator data written to the "dsp32.byt" file.)

To exit the simulator, type:

```
quit <car ret>
```

4) Run the GETSIM program; type:

```
GETSIM <car ret>
```

GETSIM expects a file with the name of "dsp32.byt" although this may be changed by altering the source for GETSIM, getsim.pas. Of course, step 3 above must use the new filename as well.

NOTE: Step 4 converts "DSP32.BYT" from an ASCII file to a binary file and saves it under the name "DSP32.BIN". Any method may be used to convert the simulator output to a binary file of consecutive program op-code bytes; the GETSIM program is just one way.

---

5) Copy the DSP32.BIN file to the directory with Hypersignal software. Hypersignal DSP software will use this file when executing DSP32 functions.

NOTE: The GETPRG program reads program code directly from DSP32 board memory and stores it in the file "DSP32.BIN". The source for GETPRG, "GETPRG.PAS", is included on the Hypersignal DSP32 DSP Source

Code Interface diskette and may be modified to change the number of bytes written out, etc. This represents an alternative method to the use of the simulator as described above; for example, the DSP3EMU emulator is used to load the program (created by the DSP3MAKE utility) into DSP32 board memory, then the GETPRG program is run after exiting the emulator, generating a DSP32.BIN file for use by Hypersignal.

Additional Information

DSP32 Source Code Interface Modification Notes

Related Topics

AT&T WE DSP32 Source Code Interface for Hypersignal

DSP32 Source Code Interface File List for Hypersignal

DSP32 Source Code Modification Process for Hypersignal

DSP32 Source Code Interface Modification Notes

1) When you add code to the DSP32.S file, be aware of the "0x1000" boundary. Hypersignal-Macro high-level code assumes that the start of various data buffers is 0x1000, so code that lands beyond this value could be overwritten. You can check the "DSP32.L" list file The "buf0_adr" variable (location 0x1A) can be changed to instruct the high-level software to make more room for program code. However, a better approach, if possible, might be to temporarily

de-activate some functions to make room for additional code, so that the amount of data space beyond 0x1000 can be preserved.

2) If you need to use an external sampling clock, follow the basic guidelines below:

Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.

Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling

circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field). Note that a maximum value of 65536 can be represented, because integer values on the DSP32 are 16-bits.

Enter SAMPLING FREQ field values in all high-level menus to match the actual

external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

AT&T WE DSP32 Source Code Interface for Hypersignal

DSP32 Source Code Interface File List for Hypersignal

DSP32 Source Code Modification Process for Hypersignal

5.3 Motorola DSP5600x Source Code Interface

Main Topics:

Related Topics:

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface Modification Notes

5.3.1 Motorola DSP5600x Source Code Interface for Hypersignal

The Hypersignal DSP5600x DSP Source Code Interface distribution diskette contains all Motorola DSP5600x source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic DSP5600x source code includes analog I/O interrupt service routines for the LSI/Spectrum, Motorola ADS (with external ADC56000 or 56ADC16 analog I/O options), Ariel PC-56 and DSP-56 boards, and the Techron TEF-20 system.

As is, the DSP5600x source code is written in modular form, and supports the basic "modes" of operation below:

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the DSP5600x board and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.3.1 above, Motorola DSP5600x Source Code Interface for Hypersignal) that the routine supports, for example "ISR0" for operating mode zero. Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific

sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the "M56DEF" (non-executable definitions) and "M56BEG" modules, depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

DSP5600x Source Code Interface Host PC Communication

If the user wishes to alter or add code functionality, or improve the functionality and/or performance of existing code, he/she must keep in mind the necessary intercommunication variables between Hypersignal software and DSP5600x code. The two-way communication between the host-PC and DSP5600x set up in the Hypersignal software must not be changed, as the software expects to read/write certain memory

locations. These memory locations reside at locations "intercom" and "intermod" and are clearly commented inside M56DEF. *Do not insert any code of any type that would change the current addresses of these variables*. If necessary, you may add further communication variables for your own high-level code. The values of ANABFK, WNDBFK, RIBUFK, and MPBUFK (time data buffer, FFT window data buffer, real/imaginary data buffers, & magnitude/phase data buffers, respectively) can be changed if it becomes necessary to alter memory allocation in your DSP5600x code and communicate these changes to the host PC). The DSP5600x source code has been well-documented to aid the user in these areas.

For all DSP5600x boards, you should avoid using the program memory area between 20H and 0FFH, which is used by Hypersignal to implement a host-interface protocol (20H-3FH), store host-DSP communication variables and structures (40H-7FH) and local DSP variables (80H-8FH). Also, note that for some DSP5600x boards, Hypersignal performs a 'pseudo-reset', in which a host interface vector is used to reset the chip, peripherals, and interrupts, and then jump to the actual reset vector at location zero.

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

DSP5600x Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground.

Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5", for a good example of a real-time, interrupt-driven structure.

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

5.3.2 DSP5600x Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal DSP5600x DSP Source Code Interface distribution diskette:

variable & constant definition module	
reset and interrupt vector locations, and host interface	
initialization, including variables, operating modes, interrupt vectors, and analog I/O drivers	

continuous A/D ISR for all boards

continuous D/A ISR for all boards

Digital Oscilloscope ISR for all boards

Spectrum Analyzer ISR for all boards

Stimulus & Response ISR for all boards

foreground A/D-to-disk processing (mode=0)

foreground D/A-from-disk processing (mode=1)

foreground Digital Oscilloscope processing (mode=2)

FFT file-based acceleration processing (mode=3)

iFFT file-

based
acceleration
processing
(mode=4)

foreground
Spectrum
Analyzer
processing
(mode=5)

foreground
stimulus &
response
processing
(mode=6)

convolution/
correlation
file-based
acceleration
processing
(mode=7)

arctangent
routine

convolution/ correlation routine

fractional and integer divide routines

exponentiation (base-2) routine (needed for fixed-point transfer function calculation)

complex FFT/iFFT routine of variable length N

(N=2^{m}**m)

filter control routine for real-time filtering

FIR filter routine

IIR (cascade sections, modified direct form) filter routine

IIR (cascade sections, balanced pole-zero form) filter routine

log2 approximation routine

magnitude routine (complex input)

4-quadrant phase routine (–π to +π; –pi to +pi, complex input)

post-FFT processing (mag, phase, scaling)

post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation

squareroot

routine

window routine: applies PC-downloaded window for Spectrum Analyzer and FFT

complex-ratio routine used for transfer function

location of maximum value in X:buffer data

location of minimum value in X:buffer data

location of maximum value in Y:buffer data

move routines between X: and Y: memory

equate file
for I/O
locations

hardware
equate file
for TBS
board only

master linker
control file

another linker control file, containing memory and I/O access wait-state value for all boards execpt TBS

another linker control file, containing memory and I/O access wait-state value for all TBS boards

batch file that can be used to assemble individual source code modules (use in form "ASM filename")

batch file that performs linking (use in form

"LNK")

The distribution disk does not initially contain all .CLN files necessary to produce a complete baseline, default executable code module when the M56LINK.CTL control file (above) is invoked with the Motorola DSPLNK utility. Each .ASM module above must be assembled first.

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

5.3.3 DSP5600x Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of DSP56001.OUT (and DSP56TBS.OUT, if you are using a Turtle Beach Systems board) before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to DSP5600x source code, keeping in mind the necessary links between Hypersignal and the DSP5600x (see discussion in DSP5600x Source Code Interface Host PC Communication, above, and M56DEF.ASM module, which is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .LNK (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (entering an ".ASM" extension after "filename" is optional). This file executes a

```
asm56000 -B -L %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the M56LINK.CTL control file (use an ASCII text-editor to update the M56LINK.CTL file).

3) Re-link all DSP5600x modules using the Motorola DSPLNK linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .LNK files. LNK.BAT executes a

```
dsplnk -Z -RM56MEM.CTL -
BDSP56001.OUT -MM56LINK.MAP -
FM56LINK.CTL
```

statement, where "M56LINK.CTL" is an ASCII linker-control file that specifies all .LNK files to be linked by the DSPLNK.EXE program. Note that all executable code that is re-locatable starts at after the M56VEC and M56DEF modules, which contain some absolutely located data (which is concerned with host communications). All code has been tested using version 4.1 of Motorola's ASM56000 and DSPLNK.

4) Copy the resulting "DSP56001.OUT" (or DSP56TBS.OUT, if you are using a Turtle Beach Systems board) file to directory with Hypersignal, re-invoke the Hypersignal software, and run the appropriate function(s). Hypersignal reads the ".OUT" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

5) If you prefer, change the lnk.bat file to produce some other "filename.out" file, instead of using the same name as the Hypersignal default. You can instruct Hypersignal to use non-default filenames in the System Config menu (see DSP Source Code Access And Control From Hypersignal, in section 5.0.2, DSP Source Code Interfaces for Hypersignal, above, for details).

Additional Information:

DSP5600x Source Code Interface Modification Notes

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

DSP5600x Source Code Interface Modification Notes

1) If you need to change default buffer addresses, please consult the comments and suggestions in the M56DEF.ASM file.

2) The LNK.BAT file produces a "M56LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series

Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

3) NOTE: The R7 and M7 registers are dedicated to the Hypersignal-Macro host-interface, and should not be used in user-defined code, for any reason, ever. This may cause a problem if you are using a C-compiler. If you need to free up the n7 register set, we suggest that you expand the fast interrupts used in the Hypersignal-Macro host-interface definition to long interrupts, and save registers used temporarily on the stack. This will result in a performance decrease, but may be necessary in order to take advantage of the development time saved by using a high-level language to program the DSP5600x.

4) If you need to use an external sampling clock, follow the basic guidelines below:

Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.

Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to

set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based

on what is entered in the current menu (SAMPLING FREQ fields).

Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

Motorola DSP5600x Source Code Interface for Hypersignal

DSP5600x Source Code Interface Interrupt Service Routines

DSP5600x Source Code Interface Host PC Communication

DSP5600x Source Code Interface Real-Time Processing Structure

DSP5600x Source Code Interface File List for Hypersignal

DSP5600x Source Code Modification Process for Hypersignal

5.4 AT&T WE DSP32C Source Code Interface

Main Topics:

Related Topics:

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface Modification Notes

5.4.1 AT&T WE DSP32C Source Code Interface for Hypersignal

The Hypersignal DSP32C DSP Source Code Interface distribution diskette contains all AT&T WE DSP32C source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic DSP32C source code includes analog I/O interrupt service routines for the Ariel DSP-32C and PC-32C (with various analog I/O options), AT&T, CAC (both AT-DSP32C and AC5 series), Burr-Brown (Intelligent Instrumentation) ZPB34, MTT Lory DSP4200, LSI/Spectrum, National Instruments AT-DSP2200, and Signalogic Sig32C series boards.

As is, the DSP32C source code is written in modular form, and supports the basic "modes" of operation below:

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the DSP32C boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating

mode (see section 5.4.1 above, AT&T WE DSP32C Source Code Interface for Hypersignal) that the routine supports, for example "ISR0" for operating mode zero. Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization ("itable" and PCW register) in RESET.S and W32BEG.S to locate the correct interrupt service routine for a particular function.

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

DSP32C Source Code Interface Host PC Communication

If the user wishes to alter or add code functionality, or improve the functionality and/or performance of existing code, he/she must keep in mind the necessary intercommunication variables between Hypersignal software and

DSP32C code. The two-way communication between the host-PC and DSP32C device set up in the Hypersignal software must not be changed, as the software expects to read/write certain memory locations. These memory locations reside at locations "intercom" and "intermod" and are clearly commented inside W32DEF. *Do not insert any code of any type that would change the current addresses of these variables*. If necessary, you may add further communication variables for your own high-level code. The values of ANABFK, WNDBFK, RIBUFK, and MPBUFK (time data buffer, FFT window data buffer, real/imaginary data buffers, & magnitude/phase data buffers, respectively) can be changed if it becomes necessary to alter memory allocation in your DSP32C code and communicate these changes to the host PC). The DSP32C source code has been well-documented to aid the user in these areas.

For all DSP32C boards, you should avoid using the program memory area between 100H and 2FFH, which is used by Hypersignal to store host-DSP interface variables and structures (100H-1FFH) and local DSP variables (200H-3FFH). Also, note that location 0FCH is used to store the page register value referred to above. This value is available for use; you can use it to ensure that your code attempts to execute in a memory area that physically exists on the DSP32C board.

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

DSP32C Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground.

Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5", for a good example of a real-time, interrupt-driven structure.

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

5.4.2 DSP32C Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal DSP32C DSP Source Code Interface distribution diskette:

reset vector, wait-state, and "page register" initialization

variable & constant definition module

initialization, including ISR vectors

continuous A/D ISR for all DSP32C boards

continuous D/A ISR for all DSP32C boards

Digital Oscillscope ISR for all DSP32C boards

Spectrum Analyzer ISR for all DSP32C boards

stimulus & response ISR for all DSP32C boards

analog I/O output ISR for special-case boards only

foreground A/D to disk processing (mode=0)

foreground D/A
from disk
processing
(mode=1)

foreground
Digital
Oscilloscope
processing
(mode=2)

FFT file-based
acceleration
processing
(mode=3)

iFFT file-based
acceleration
processing
(mode=4)

foreground
Spectrum
Analyzer
process
(mode=5)

foreground stim & response process (mode=6)

convolution/correlation file-based acceleration (mode=7)

arctangent routine

convolution/correlation routine

divide routine

complex FFT/iFFT routine of

variable length N (N=2**m)

post-FFT processing (mag, phase, etc.)

filter control routine for real-time filtering

FIR filter routine

IIR (cascade sections, modified direct form) filter routine

IIR (cascade sections, balanced pole-zero structure) filter routine

log (base 10) approximation routine

find maximum-in-buffer routine

find minimum-in-buffer routine

magnitude routine (complex input)

4-quadrant phase routine (−π to +π; −pi to +pi, complex input)

post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and

data manipulation	
squareroot routine	
apply pre-defined window (used with FFTs)	
complex-ratio routine used for transfer function	
batch file used to assemble any source module: type "ASM filename" (no extension needed)	
batch file used to link all source modules: type "LNK"	
link control file (command file for linker)	

The distribution disk does not initially contain all .O files necessary to produce a complete baseline, default executable code module when the W32LINK.CTL control file (above) is invoked with the AT&T d3ld utility. Each .S module above must be assembled first.

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

5.4.3 DSP32C Source Code Modification Process

NOTE: Be sure to save a backup copy of DSP32C.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to DSP32C source code, keeping in mind the necessary shared variable links between Hypersignal and the DSP32C (see discussion in DSP32C Source Code Interface Host PC Communication, above, and W32DEF.S module, which is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .O (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (*do not enter a ".S" extension after "filename"*). This file executes a

```
d3as -Q -l %1.s
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the W32LINK.CTL control file (use an ASCII text-editor to update the W32LINK.CTL file).

3) Re-link all DSP32C modules using the AT&T DL3D linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .O files. LNK.BAT executes the following statements:

```
D3LD -a -m -o DSP32C.OUT
W32LINK.CTL > W32LINK.MAP
@COPY W32LINK.MAP+EOF
W32LINK.MAP
@D3NM -f DSP32C.OUT >> TEMP
@COPY W32LINK.MAP+TEMP
W32LINK.MAP
@DEL TEMP
```

Additional Information:

DSP32C Source Code Interface Modification Notes

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

DSP32C Source Code Interface Modification Notes

1) Hypersignal DSP32C code uses r19 to store a "page register value" (initially written by the high level software to DSP32C onchip RAM address 00FCH) so that Hypersignal DSP32C code can execute on different DSP32C boards that use different memory modes (which sometimes causes instructions to be in different 64K segments). *You should not change r19 at any time, for any reason.*

2) Hypersignal DSP32C code uses r18 as a system stack pointer for both foreground and interrupt processing code. When storing CAU registers on the "stack", use the following sequence:

```
*r18++ = rN1e
*r18++ = rN2e
          -
          -
          -
*r18++ = rNNe
```

When restoring CAU registers from the stack, use the following sequence:

```
rN1e = r18 - 4
rNNe = *rN1--
          -
          -
          -
rN2e = *rN1--
rN1e = *rN1
r18e = r18 – K                /* K = 4 x (number of registers stored) */
```

There are three remarkable things about this process. First, the stack is always expected to increment and decrement in 4-byte chunks (always save/restore extended registers). Second, the stack is always expected to point to the *next available (free) location*. Third, when restoring, you can't simply do an r18e = r18 – 4 instruction and begin popping rXX = *r18-- instructions, because an interrupt might occur sometime after the r18 – 4 instruction and before you finish popping; the interrupt routine, if it also used r18 as a stack pointer (which all types of routines

should be able to do of a legitimate stack pointer), might then clobber a stored register (or more than one) that hadn't been popped yet.

3) Since AT&T dedicates r20-r22 to special purposes, and Hypersignal DSP32C code uses r19 and r18 as system memory-page register and stack pointer, respectively, this means that *r18-r22 are reserved in the Hypersignal DSP32C source code and should not be used arbitrarily for any reason, ever*.

4) Hypersignal DSP32C source code is built with the "ASM" and "LNK" batch files, which invoke the AT&T "d3as" and "d3ld" assembler and linker utilities, respectively. By examining the ASM and LNK batch files, you should be able to construct an equivalent method of building the code using the AT&T "make" utility instead; however, producing a memory map file will be difficult. Obscure DOS commands have been used to create a map file in the LNK.BAT file, since the AT&T utilities will only print map output to the screen. The map file produced has a filename of "W32LINK.MAP"; you can use this file to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the

executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

5) Because the different DSP32C boards supported use different memory modes, Hypersignal DSP32C source code does not specify any memory mode of operation. In fact, the "LNK" batch file invocation of the AT&T linker (d3ld) does not specify any memory mode or configuration, causing the linker to assume contiguous memory starting at zero. This is possible because:

The Hypersignal DSP32C source code always assumes at least 800H of read/write memory available at location zero (true for all memory modes). This area is used for the reset vector, storage of the "page register" value, storage of shared variables between host PC and DSP32C, storage of most DSP32C local variables, and system stack.

Use of pc-relative "pccall" and "pcgoto" instructions that interpret procedure entry points and branch destinations as *offsets from the current pc (program counter)*

value, rather than absolute constants.

Use of r19 as a pre-defined "page register" for setting the upper 8 bits of registers being used to access local variables and interrupt table vectors not inside the 0-7FFH memory area. See discussion in 1), above.

6) The DSP32C apparently has a bug in the "call(rNN)" instruction in that only the least-significant 16 bits of the current program counter value are stored in rNN before the branch is taken. This is why all Hypersignal DSP32C code "pccall(rNN)" instructions perform a rNNe = rNN | r19 (logical "OR" with r19) as their latent instruction. We highly recommend that any calls you add to the source code do likewise.

7) In addition to setting them directly in the DSP32C source code, IOC register settings can be controlled from the high-level software in order to accommodate non-standard analog I/O connected to DSP32C serial ports. Inside the Hypersignal-Macro configuration file there is a 32-bit integer IOC value for every DSP32C board currently supported by Hypersignal-Macro--see the "CONFIG.DOC" file on Hypersignal distribution diskette #2 for a list of the sequence of variables in the configuration file. Depending on the board currently in use, one of these values is sent down to the DSP32C board whenever a Hypersignal-Macro

real-time function requiring analog I/O is invoked. The corresponding DSP source code variable is "IOC_reg", which is normally used directly to set the IOC register (unless changed by the user in the DSP source code).

8) If you need to use an external sampling clock, follow the basic guidelines below:

Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.

Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value

correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all

real-time and post-acquisition displays are accurate.

Related Topics

AT&T WE DSP32C Source Code Interface for Hypersignal

DSP32C Source Code Interface Interrupt Service Routines

DSP32C Source Code Interface Host PC Communication

DSP32C Source Code Interface Real-Time Processing Structure

DSP32C Source Code Interface File List for Hypersignal

DSP32C Source Code Modification Process for Hypersignal

5.5 Texas Instruments TMS320C3x Source Code Interface

Main Topics:

Related Topics:

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface Modification Notes

5.5.1 Texas Instruments TMS320C3x Source

Code Interface for Hypersignal

The Hypersignal TMS320C3x DSP Source Code Interface distribution diskette contains all Texas Instruments TMS320C3x source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic TMS320C3x source code includes analog I/O interrupt service routines for the LSI/Spectrum system, ASPI Banshee, Sonitech Spirit-30, DSP Research TIGER 30, MTT Lory DSP4100, Domain Technologies DSPCard, Innovative Integration PC31, and Texas Instruments EVM boards.

As is, the TMS320C3x source code includes both library routines and also real-time interrupt driven "modes" of operation that use foreground processing and interrupt service routine processing to implement standard Hypersignal functions. These functions have been written in modular form, and support the basic "modes" of operation below:

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the TMS320C3x boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.5.1 above, TMS320C3x Source Code Interface for Hypersignal) that they

support, for example "ISR0". Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the initial setup code in "T30BEG", depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

TMS320C3x Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she must keep in mind the necessary links between Hypersignal software and the TMS320C3x code. The two-way communication between the PC and TMS320C3x processor established in the Hypersignal

software must not be changed, as the high-level software expects to read/write certain memory locations. These memory locations are initialized by the high-level software before any code executes on the DSP board (some locations are actually read by the high-level software as well, allowing you to change such "constants" as buffer base addresses). *Do not insert any code of any type before these addresses*. The TMS320C3x source code has been documented to aid the user in these areas.

For all TMS320C3x boards, you should avoid using the program memory areas between 0CH and 0C0H, which is reserved by Texas Instruments, between 0C0H and 400H, which is often used for debuggers and monitors (for Texas Instruments EVM boards, Hypersignal uses 300H to 3FFH for a host interace), between 400H and 47FH, which is used by Hypersignal to store host-DSP interface variables and structures, and between 480H and 4FFH, which is used by Hypersignal TMS320C3x DSP code to store local variables.

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

TMS320C3x Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. The dual-buffer communication between the host and DSP is fully synchronized using "interlock" buffer flags. Use mode 5 source modules ("MODUL5" and routines it calls) and "xxx_ISR5" for one example of a real-time, interrupt-driven structure.

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

5.5.2 TMS320C3x Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal TMS320C3x DSP Source Code Interface distribution diskette:

variable & constant definition module	
initialization, including ISR vectors	
continuous A/D ISR for all boards	
continuous D/A ISR for all boards	

Digital Scope ISR for all boards

Spectrum Analyzer ISR for all boards

stimulus & response ISR for all boards

foreground A/D to disk processing (mode=0)

foreground D/A from disk

processing
(mode=1)

foreground
Digital
Oscilloscope
processing
(mode=2)

FFT file-based
acceleration
processing
(mode=3)

iFFT file-based
acceleration
processing
(mode=4)

foreground
Spectrum
Analyzer

process (mode=5)

foreground stimulus & response process (mode=6)

convolution/correlation file-based acceleration (mode=7)

arctangent routine

convolution/correlation routine

floating-point
divide routine

complex
FFT/iFFT
routine of
variable length
N (N=2^{m}**m)

post-FFT
processing
(mag, phase,
etc.)

log
approximation
routine

magnitude
routine
(complex

input)

find maximum-in-buffer routine

find minimum-in-buffer routine

4-quadrant phase routine (−π to +π; −pi to +pi, complex input)

sine value lookup-table for FFT use

post-FFT processing for Spectrum Analyzer

function, including mag, phase, power-spectra, log, scaling, and data manipulation	
squareroot routine	
apply pre-defined window (used with FFTs)	
filter control routine for real-time filtering	
FIR filter routine	

IIR (cascade sections, modified direct form) filter routine	
IIR (cascade sections, balanced pole-zero structure) filter routine	
complex-ratio routine used for transfer function	
batch file used to assemble any source module: type "ASM filename" (no extension needed)	
batch file used to link all source modules: type "LNK"	

link control file (command file for linker)	

The distribution diskette does not initially contain all .OBJ files necessary to produce a complete baseline, default executable code module when the T30LINK.CMD control file (above) is invoked with the Texas Instruments LNK30 utility. Each .ASM module above must be assembled first.

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

5.5.3 TMS320C3x Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of TMS32030.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to TMS320C3x source code, keeping in mind the necessary shared variables between Hypersignal and the TMS320C3x (the T30DEF.ASM module is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .OBJ (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (entering an ".ASM" extension after "filename" is optional). This file executes a

asm30 -L %1

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the T30LINK.CMD control file (use an

ASCII text-editor to update the T30LINK.CMD file).

3) Re-link all TMS320C3x modules using the Texas Instruments LNK30 linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ files. LNK.BAT executes a

lnk30 T30LINK.CMD

statement, where "T30LINK.CMD" is an ASCII linker-control file that specifies all .OBJ files to be linked by the LNK30.EXE program. Note that T30LINK.CMD places shared and local variables at 400H and 480H, respectively. All code has been tested using version 4.5 of the Texas Instruments TMS320C3x/C4x development tools.

4) Copy the resulting "TMS32030.OUT" file to the directory with Hypersignal software, run Hypersignal, and invoke the appropriate functions. Hypersignal reads the "COFF" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information:

TMS320C3x Source Code Interface Modification Notes

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

TMS320C3x Source Code Modification Process for Hypersignal

TMS320C3x Source Code Interface Modification Notes

1) Avoid using the default ".TEXT", ".DATA", or ".BSS" sections in any new code you might add. For some reason, the Texas Instruments linker wants to put these sections in front of all named sections, unless you specifically tell it otherwise in the .CMD file. If one of these sections loads in front of the "INIT" section, which is supposed to load at zero because it contains the reset vector, your code is dead.

2) *Do not declare new variables before or in the "HOST_COMM" section (file T30DEF.ASM).* These are the host-DSP intercommunication memory areas that must remain unchanged to allow normal Hypersignal operation.

3) Note that the start of executable code used by Hypersignal is at 500H--if you declare enough variables in section "LOCAL" (file T30DEF.ASM) or immediately after it, you could overrun the start address. If necessary put the variables somewhere else or change

the address of "INIT" (both in T30BEG.ASM and T30LINK.CMD). Also note that analog and acceleration processing data buffers start at address 2000H. If you add enough code, you could overrun this address. If so, change the buffer address value "ANABFK" only--Hypersignal will base all other buffers on this value. *If you need to move any buffer addresses, please see the T30DEF.ASM file--it contains memory maps and numerous suggestions on changing default addresses.*

4) For ASPI Banshsee boards, any variables between 400H and 500H (normally includes all shared and local variables) are translated to dual-port memory area 800400H-8004FFH by the Hypersignal high-level downloader, and the "VARS" location is written as 80H. For the LSI board the 400H-4FFH area is translated to dual-port memory area 30400H, and the "VARS" location is written as 3. These relocations are done so that shared variables can be accessed by the host without putting the processor in hold.

5) For the LSI board, data buffers are also moved to the dual-port memory area (instead of at 2000H, in the "program" memory area). See the T30DEF.ASM module for details on the exact translation that occurs and under what circumstances.

6) The LNK.BAT file produces a "T30LINK.MAP" map file which you can use to check for overlaps

and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

7) If you need to use an external sampling clock, follow the basic guidelines below:

Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.

Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of

the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is

entered in the current menu (SAMPLING FREQ field).

Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

SPECIAL NOTES FOR THE TEXAS INSTRUMENTS EVM BOARD AND DOMAIN TECHNOLOGIES DSPCARD

The Texas Instruments EVM board and Domain Technologies DSPCard are special cases. For both these boards, the .ASM files EVM_INIT, GET_COMM, RD_DATA, and WR_DATA can be assembled and linked (using the EVMLNK.BAT and DOMLNK.BAT files) to produce the small "host interface" executable programs "EVM_INIT.OUT" and "DOM_INIT.OUT". These files must be downloaded and run by the Texas Instruments EVMLOAD program or Domain Technologies LDR_C30 program, respectively, prior to running Hypersignal. You can use DOS commands of the following form:

```
EVMLOAD   EVM_INIT.OUT
<car ret>
```

or

```
LDR_C30   DOM_INIT.OUT
<car ret>
```

The EVMLOAD.EXE file should be included with your EVM board from Texas Instruments. *You must enter the ".OUT" extension when using the EVMLOAD program. Also, version 1.51 and above of EVMLOAD requires the two files "RESVCT.001" and "RESVCT.002" to be present in order to run--otherwise it spits up an "unable to initialize emulator" message*. A default "EVM_INIT.OUT" file is included on Hypersignal distribution diskette #2. Once loaded, the host interface provides a mechanism for Hypersignal to download further code and real-time data, as necessary for standard operation.

The Domain LDR_C30.EXE program can be found on the Domain "Development System for Texas Instruments 320C3x" disk. A default DOM_INIT.OUT file is included on Hypersignal distribution diskette #2.

The reason for separate host-interface files (as opposed to including the host interface section inside the standard TMS32030.OUT file) is that Texas Instruments apparently regards the downloading of code directly through the Test Bus Controller (TBC) embedded-emulation interface on the EVM board as

proprietary (even though we asked for this information, they declined to provide it). Obviously, the use of separate .OUT files for programs that must coexist (and execute) at the same time can create problems. Below is some discussion of this situation and things to watch out for should you desire to use the Hypersignal DSP Source Code Interface to change TMS320C3x source code for either EVM or DSPCard operation.

The EVM_INIT.OUT and DOM_INIT.OUT code files expect to start execution at 300H by the. The area from 300H to 400H should be represented by an "uninitialized section" (as it is in the default Hypersignal TMS32030.OUT file), preventing the Hypersignal COFF file loader (or loaders from other vendors) from overwriting the EVM or DSPCard host interfaces, once present in memory and running.

When Hypersignal puts the EVM board or DSPCard in "reset", what it is actually doing is changing the reset vector to point to an idle address, where the TMS320C30 or TMS320C31 spins while still allowing the interrupt-driven host interface to function. When Hypersignal "runs" the EVM board or DSPCard, the reset vector is changed again to allow normal or user-defined functions to execute (using the reset vector in the TMS32030.OUT file). Hypersignal never puts the EVM TMS320C30 or DSPCard

TMS320C31 chips in hold, since to do so would not allow any data transfer while in the hold state.

Should you get in to the technical details of it, you will find that the Hypersignal version of the EVM host interface is much less complicated than the example interface provided by Texas Instruments. No elaborate command formatting or parameter area is required. Since commands can be byte-wide, the Hypersignal interface simply sends the command type, instead of a "command start" bit followed by parameters and a "command end" bit. In the Hypersignal interface, the starting address is simply a pre-defined data transfer of one command type; other command types delineate the basic transfer mode (read 16-bit data, read 32-bit data, write 16-bit data, etc.). The count parameter has been eliminated completely since the EVM host interface always auto-increments its transfer address, and the host software always controls the number of transfers.

Related Topics

Texas Instruments TMS320C3x Source Code Interface for Hypersignal

TMS320C3x Source Code Interface Interrupt Service Routines

TMS320C3x Source Code Interface Host PC Communication

TMS320C3x Source Code Interface Real-Time Processing Structure

TMS320C3x Source Code Interface File List for Hypersignal

5.6 NEC 77220 Source Code Interface

Main Topics:

Related Topics:

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface Modification Notes

5.6.1 NEC 77220 Source Code Interface for Hypersignal

The Hypersignal 77220 DSP Source Code Interface distribution diskette contains all NEC 77220 source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic NEC 77220 source code includes analog I/O routines for the NEC DDK77220 board.

As is, the 77220 source code includes both library routines and also real-time interrupt driven "modes" of operation that use foreground processing and interrupt service routine processing to

implement standard Hypersignal functions. These functions have been written in modular form, and support the basic "modes" of operation below:

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface File List for Hypersignal

77220 Source Code Modification Process for Hypersignal

77220 Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she

must keep in mind the necessary links between Hypersignal software and the 77220 code. The two-way communication between the PC and 77220 established in the Hypersignal software must not be changed, as the high-level software expects to read/write certain memory locations. These memory locations are initialized by the high-level software before any code executes on the DSP board (some locations are actually read by the high-level software as well, allowing you to change such "constants" as buffer base addresses). *Do not insert any code of any type that will disturb these addresses*. The 77220 source code has been documented to aid the user in these areas.

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface File List for Hypersignal

77220 Source Code Modification Process for Hypersignal

77220 Source Code Interface Real-Time Processing Structure

The 77220 only has one external interrupt (except for NMI) which is currently used to implement a 'pseudo-hold' function to facilitate communication with the host PC (see 77220 Source Code Interface Modification Notes, below). Thus,

for the DDK 77220 board implementation, all analog I/O is performed via polling of the onchip serial port status, requiring that all analog I/O data processing be done in the "foreground", on a buffer-loop basis. Since further processing on buffers must begin as soon as they are full, this means that if the processing to be done is significant (which it is for the Spectrum Analyzer function), then the next data buffer won't be continuous, because no further samples can be acquired (without a sample interrupt) until the current buffer is completely processed.

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface File List for Hypersignal

77220 Source Code Modification Process for Hypersignal

5.6.2 77220 Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal 77220 DSP Source Code Interface distribution diskette:

variable & constant definition module

initialization, including ISR vectors

foreground A/D to disk processing (mode=0)

foreground D/A from disk processing (mode=1)

foreground Digital Oscilloscope processing (mode=2)

FFT file-based acceleration processing (mode=3)

iFFT file-based acceleration processing (mode=4)

foreground Spectrum Analyzer process (mode=5)

foreground stimulus & response processing (mode=6)

arctangent routine

integer-divide routine

complex FFT/iFFT routine of variable length N (N=2**m)

post-FFT processing (mag, phase, etc.)

log approximation routine

magnitude routine (complex input)

find maximum-in-buffer routine

find minimum-in-buffer routine

4-quadrant phase routine (−π to +π; −pi to +pi, complex input)

sine value lookup-table for FFT use

post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation

squareroot routine

apply pre-defined window (used with FFTs)

filter control routine for real-time filtering

FIR filter routine

IIR (cascade sections, modified direct form) filter routine

IIR (cascade sections, balanced pole-zero structure) filter routine

include file containing reference to external variable definitions for use in other .SRC files (see N20DEF.SRC above)

batch file used to assemble any source module: type "SRC filename" (no extension

needed)

batch file used to assemble any assembly module: type "ASM filename" (no extension needed)

batch file used to link all source modules: type "LNK"

link control file (command file for linker)

The distribution diskette does not initially contain all .REL files necessary to produce a complete baseline, default executable code module when the N20LINK.CTL control file (above) is invoked with the NEC LK77230 utility. Each .SRC module above must be assembled first.

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

5.6.3 77220 Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of NEC77220.HEX before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to 77220 source code, keeping in mind the necessary shared variables between Hypersignal and the 77220 (the N20DEF.SRC module is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .OBJ (object) file is generated for each module. For convenience, a "SRC.BAT" batch file has been provided that you may use in the form of:

```
SRC filename <car ret>
```

to assemble source code modules (an ".SRC" extension after "filename" is optional). This file executes a

```
ra77230 nopre=%1 cp=20
object=%1 pri=%1.LST
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the N20LINK.CTL control file (use an ASCII text-editor to update the N20LINK.CTL file).

3) Re-link all 77220 modules using the NEC 77220 linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .LNK files. LNK.BAT executes the following statements:

```
lk77230 !N20LINK.CTL oc77230
NEC77220.LNK to NEC77220.TMP
hibyt220
NEC77220.TMP NEC77220.HEX del
NEC77220.TMP
```

where "N20LINK.CTL" is an ASCII linker-control file that specifies all .LNK files to be linked by the LK77230.EXE program. Note that N20LINK.CTL blocks out (reserves) memory from 1000H to 13FFH for interrupt handling, shared and local variables, and initialization code. Thus, all relocatable code starts at 1400H. All code has been tested using version 2.0 of the NEC 77230/77220 development tools.

4) Copy the resulting "NEC77220.HEX" file to directory with Hypersignal software, run Hypersignal, and use appropriate functions. Hypersignal reads the "Intel hex" file format directly, downloads all code and data, and

begins execution at the reset vector (location zero).

Additional Information:

77220 Source Code Interface Modification Notes

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface File List for Hypersignal

77220 Source Code Modification Process for Hypersignal

77220 Source Code Interface Modification Notes

1) Although hard-core NEC 77220 programmers might say that the 77220 assembly language (".ASM" files) is the only way to go, our programmers, with a great deal of experience in programming many other DSP devices, felt that the "pre-assembly" language (".SRC" files) was much more like other DSP assembly languages and thus made creation of the 77220 source code more standard and easy-to-read. We have found the pre-assembler to do an excellent job in organizing code to take advantage of pipeline "delay tricks" in the 77220, as well as reliable in ensuring that either a NOP or non-related instruction is inserted when the pre-assembly source calls for use of a result or register too early. Since the pre-assembler can output

its equivalent ".ASM" file at any time for manual optimization by the user, we feel that the development time saved by the pre-assembler more than justifies its use. Thus, all Hypersignal 77220 routines are ".SRC" files.

2) Note that the "N20LINK.CTL" file (command file for linker) blocks out fast instruction memory from 0 to 3FFH. This is because these areas are occupied by absolute sections: "RESET" at 0, "INT" (interrupt service routine) at 100H, "SHARED_VARS" and "LOCAL_VARS" at 200H and 280H, respectively, and "INIT" at 300H. Thus, all relocatable modules start at 400H, and are organized into successive chunks as long as necessary by the order in which they appear in the ".CTL" file.

3) *Do not declare new variables before or in the "SHARED_VARS" section (file N20DEF.SRC) starting at 200H in fast instruction memory*. These are the shared host-DSP intercommunication memory areas that must remain unchanged to allow normal Hypersignal operation. To add memory variables, we suggest adding them after existing variables in the "LOCAL_VARS" section (also file N20DEF.SRC) starting at 280H. Note that these sections reserve memory from 200H to 2FFH in fast instruction memory, and that there is code both before and after this section, so be careful of adding too many variables.

4) The LNK.BAT file produces a "N20LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

5) The "pseudo-hold" interrupt routine at 100H responds to a "HOLD" interrupt from the host PC, saves any of the current machine state that might be modified, and calls a routine in internal instruction EPROM that waits until the host PC says it is "free to go." This effectively puts the 77220 in a hold state, allowing the host PC to access the external data/instruction memory on the board. This mechanism allows the high-level Hypersignal software to operate the DDK 77220 board in a real-time mode, transferring analog I/O data continuously when necessary, and also allows the high-

level software and the low-level 77220 code to operate asynchronously using "message passing" without the need to reset the 77220 or synchronize using I/O register flags.

6) If you need to use an external sampling clock, follow the basic guidelines below:

Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.

Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you

introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and

post-acquisition displays are accurate.

Related Topics

NEC 77220 Source Code Interface for Hypersignal

77220 Source Code Interface Host PC Communication

77220 Source Code Interface Real-Time Processing Structure

77220 Source Code Interface File List for Hypersignal

77220 Source Code Modification Process for Hypersignal

5.7 Analog Devices ADSP-21xx Source Code Interface

Main Topics:

Related Topics:

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface Modification Notes

5.7.1 Analog Devices ADSP-21xx Source Code Interface for Hypersignal

The Hypersignal ADSP-21xx DSP Source Code Interface distribution diskette contains all Analog Devices ADSP-21xx source code for ADSP-2100, ADSP-2101 and ADSP-2115 boards, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic ADSP-21xx source code includes analog I/O interrupt service routines for the LSI/Spectrum 2100 and 2101 system boards.

As is, the ADSP-21xx source code includes both library routines and also real-time interrupt driven "modes" of operation that use foreground processing and interrupt service routine processing to implement standard Hypersignal functions. These functions have been written in modular form, and support the basic "modes" of operation below:

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on ADSP-21xx boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.7.7 above, ASDP-21xx Source Code Interface for Hypersignal) that they support, for example "ISR0". Inside each

interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the initial setup code in "A21BEG", depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

ADSP-21xx Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she must keep in mind the necessary links between Hypersignal software and ADSP-21xx code. The two-way communication between the PC and ADSP-21xx processor established in the Hypersignal software must not be changed, as

the high-level software expects to read/write certain memory locations. These memory locations are initialized by the high-level software before any code executes on the DSP board (some locations are actually read by the high-level software as well, allowing you to change such "constants" as buffer base addresses). *Do not insert any code of any type before these addresses*. The ADSP-21xx source code has been documented to aid the user in these areas.

For all ADSP-21xx boards, you should avoid using the program memory areas between 1CH and 07FH, which is used by Hypersignal to store host-DSP interface variables and structures, and between 80H and 8FH, which is used by Hypersignal ADSP-21xx DSP code to store local variables. Note also that if Hypersignal ADSP-21xx DSP code is allowed to run, filter coefficients may be stored in onchip RAM at location 3C00H.

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

ADSP-21xx Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. The dual-buffer communication between the host and DSP is fully synchronized using "interlock" buffer flags. Use mode 5 source modules ("MODUL5" and routines it calls) and "xxx_ISR5" for one example of a real-time, interrupt-driven structure.

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

5.7.2 ADSP-21xx Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal ADSP-21xx DSP Source Code distribution diskette:

variable & constant definition module

power-on/reset initialization

interrupt and reset vector definition

continuous

A/D ISR for LSI/Spectrum system board

continuous D/A ISR for LSI/Spectrum system board

Digital Oscilloscope ISR for LSI/Spectrum system board

Spectrum Analyzer ISR for LSI/Spectrum system board

Stimulus & response ISR for LSI/Spectrum system board

foreground A/D to disk

processing
(mode=0)

foreground
D/A from
disk
processing
(mode=1)

foreground
digital scope
processing
(mode=2)

FFT file-
based
acceleration
processing
(mode=3)

iFFT file-
based
acceleration

processing
(mode=4)

foreground
Spectrum
Analyzer
process
(mode=5)

foreground
stimulus &
response
process
(mode=6)

convolution/c
orrelation
file-based
acceleration
(mode=7)

arctangent
routine

bit-reverse routine

convolution/correlation routine

fractional and integer divide routines

complex FFT/iFFT routine of variable length N

(N=2^{m}**m)

post-FFT processing (mag, phase, etc.)

log approximation routine

magnitude routine (complex input)

find maximum-in-buffer routine

find minimum-in-buffer routine

4-quadrant phase routine (–π to +π; –pi to +pi, complex input)

sine value lookup-table for FFT use

post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation

squareroot routine

apply pre-defined

window (used with FFTs)

filter control routine for real-time filtering

FIR filter routine

IIR (cascade sections, modified direct form) filter routine

IIR (cascade sections, balanced pole-zero structure) filter routine

complex-ratio routine used for transfer function

batch file used to assemble any source module: type "ASM filename" (no extension needed)

batch file used to link all source modules: type "LNK"

link control file (command file for linker)

"system builder" file

"system architecture" file

The distribution diskette does not initially contain all .OBJ, .CDE, and .INT files necessary to produce a complete baseline, default executable code module when the A21LINK.CTL control file (above) is invoked with the Analog Devices LD21 link utility. Each .DSP module above must be assembled first.

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

5.7.3 ADSP-21xx Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of ADSP21xx.EXE before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to ADSP-21xx source code, keeping in mind the necessary shared variables between Hypersignal and the ADSP-21xx (the A21DEF.DSP module is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .OBJ (object), .CDE, and .INT files are generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (entering a ".DSP" extension after "filename" is optional). This file executes a

```
ASM21 -L %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the A21LINK.CTL control file (use an

ASCII text-editor to update the A21LINK.CTL file).

3) Re-link all ADSP-21xx modules using the Analog Devices LD21 linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ, .CDE, and .INT files. LNK.BAT executes the following statements:

```
LD21 A21LINK.CTL -a HSMACRO -
e AD21xx -g -x
DEL A21LINK.MAP
REN AD21xx.MAP A21LINK.MAP
```

where "A21LINK.CTL" is an ASCII linker-control file that specifies all .OBJ, .CDE, and .INT files to be linked by the LD21 link utility, and "A21LINK.MAP" is a map file showing variable and module addresses and lengths (see ADSP-21xx Source Code Interface Modification Notes, below). Note that A21LINK.CTL places shared and local variables at 001CH, forcing relocatable code to start after these declarations. All code has been tested using version 1.13 (11/21/90) of the Analog Devices ASM21 assembler, and version 1.12 (11/15/90) of the Analog Devices LD21 linker.

4) Copy the resulting "ADSP21xx.EXE" file to the directory with Hypersignal software, run Hypersignal, and invoke the appropriate functions. Hypersignal reads the Analog Devices .EXE file format directly, downloads all code and data, and

begins execution at the reset vector (location zero).

Additional Information:

ADSP-21xx Source Code Interface Modification Notes

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

ADSP-21xx Source Code Interface Modification Notes

1) The basic LSI ADSP-2100 and ADSP-2101 boards support only 4kX24 of program memory. Thus, you must be careful when adding program memory variables or code so that overall code space is not exceeded. Program memory can be expanded to 16kX24.

2) Because of the 4kX24 minimum program memory size, the look-up table used by the Hypersignal FFT routine, which is stored in program memory, has been limited to a 1024-pt. FFT size. Because the FFT routine used requires the table to be in bit-reversed order, both cosine and sine values are required. Unfortunately, the ADSP-21xx bit-

reversed addressing facility is not very flexible, not only placing restrictions on where such values are located in memory, but also creating the actual bit-reversed address only in hardware without making it available in software (via register). Should you desire to increase the FFT size, you must obtain expanded memory and also create a new bit-reversed look-up table.

3) In its factory configuration, the LSI ADSP-2101 board supports only 4kX16 of data memory. Data memory can be expanded, but not to 16kX16 because the 2101 only provides 14 external address lines, and in addition, uses the upper 4kX16 of data memory space for onchip internal RAM and other chip peripherals (serial ports, etc.). Thus, the maximum available data memory size for standard Hypersignal purposes is 12kX16.

4) If you need to use an external sampling clock, follow the basic guidelines below:

Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.

Enter '-EXT' after the board designator in the Hypersignal System Configuration

menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". The value is in Hz (Hertz); note

that for ADSP-21xx boards, which use 16-bit memory accesses, the maximum value that can be represented is limited to 65535. This value is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

5) The LNK.BAT file produces a "A21LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay

extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

Related Topics

Analog Devices ADSP-21xx Source Code Interface for Hypersignal

ADSP-21xx Source Code Interface Interrupt Service Routines

ADSP-21xx Source Code Interface Host PC Communication

ADSP-21xx Source Code Interface Real-Time Processing Structure

ADSP-21xx Source Code Interface File List for Hypersignal

ADSP-21xx Source Code Modification Process for Hypersignal

5.8 Motorola DSP96002 Source Code Interface

Main Topics:

Related Topics:

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface Modification Notes

5.8.1 Motorola DSP96002 Source Code Interface for Hypersignal

The Hypersignal DSP96002 DSP Source Code Interface distribution diskette contains all Motorola DSP96002 source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic DSP96002 source code includes analog I/O interrupt service routines for the LSI/Spectrum and Ariel DSP-96 boards.

As is, the DSP96002 source code is written in modular form, and supports the basic "modes" of operation below:

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the DSP96002 board and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.8.1 above, Motorola DSP96002 Source Code Interface for Hypersignal) that the routine supports, for example "ISR0" for operating mode zero.

Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the "M96DEF" (non-executable definitions), "M96VEC", and "M96BEG" modules, depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

DSP96002 Source Code Interface Host PC Communication

If the user wishes to alter or add code functionality, or improve the functionality and/or performance of existing code, he/she must keep in mind the necessary intercommunication variables between Hypersignal software and DSP96002 code. The two-way

communication between the host-PC and DSP96002 set up in the Hypersignal software must not be changed, as the software expects to read/write certain memory locations. These memory locations reside at 100H and are clearly commented inside M96DEF. *Do not insert any code of any type that would change the current addresses of these variables*. If necessary, you may add further communication variables for your own high-level code. The values of ANABFK, WNDBFK, RIBUFK, and MPBUFK (time data buffer, FFT window data buffer, real/imaginary data buffers, & magnitude/phase data buffers, respectively) can be changed if it becomes necessary to alter memory allocation in your DSP96002 code and communicate these changes to the host PC). The DSP96002 source code has been well-documented to aid the user in these areas.

For all DSP96002 boards, you should avoid using the program memory areas between 1CH and 03FH, which is used by Hypersignal to implement a host-interface, between 40H and 0FFH, which is used by Hypersignal to implement setup and initialization code necessary to coordinate execution and DMA memory-mapped interfaces with the host, between 100H and 17FH, which is used by Hypersignal to store host-DSP interface variables and structures, and between 180H and 1FFH, which is used by

Hypersignal DSP96002 DSP code to store local variables. Note that you can take advantage of the start up code located between by 40H and 0FFH by locating the start of your executable code at 400H, which is the initial jump taken by Hypersignal DSP96002 code after all setup and communication initialization is complete.

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

DSP96002 Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts, although all ISRs save all registers temporarily used. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground.

Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5", for a good example of a real-time, interrupt-driven structure.

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

5.8.2 DSP96002 Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal DSP96002 DSP Source Code Interface distribution diskette:

variable & constant definition module

beginning vectors, including

ISR vector initialization, host-interface definition, and power-on/reset chip initialization

software initialization

continuous A/D ISR for all boards

continuous D/A ISR for all boards

Digital Oscilloscope ISR for all boards

Spectrum Analyzer ISR for all boards

stimulus & response ISR for all boards

foreground A/D-to-disk processing (mode=0)

foreground D/A-from-disk processing (mode=1)

foreground Digital Oscilloscope processing

(mode=2)

FFT file-based acceleration processing (mode=3)

iFFT file-based acceleration processing (mode=4)

foreground Spectrum Analyzer processing (mode=5)

foreground stimulus & response processing (mode=6)

convolution/
correlation
file-based
acceleration
processing
(mode=7)

arctangent
routine

convolution/
correlation
routine

fractional
and integer
divide
routines

DMA transfer preparation routine (see DSP96002 Source Code Interface Modification Notes, below)

complex FFT/iFFT routine of variable length N (N=2**m)

post-FFT processing (mag, phase, scaling)

filter control routine for real-time filtering

FIR filter routine

IIR (cascade sections, modified direct form) filter routine

IIR (cascade sections, balanced pole-zero form) filter routine

log approximation routine

magnitude routine (complex input)

4-quadrant

phase	routine (–π to +π; –pi to +pi, complex input)
post-FFT	processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation
squareroot	routine
window	routine: applies PC-downloaded window for Spectrum Analyzer and FFT
transfer	function calculation

(complex input)

location of maximum value in X:buffer data

location of minimum value in X:buffer data

location of maximum value in Y:buffer data

move routines between X: and Y: memory

include file
containing

equates for internal I/O addresses

include file containing equates for external I/O addresses

link control file (control file for linker)

batch file that can be used to assemble individual source code modules (use in form "ASM

filename")

batch file that performs linking (use in form "LNK")

The distribution diskette does not initially contain all .CLN files necessary to produce a complete baseline, default executable code module when the M96LINK.CTL control file (above) is invoked with the Motorola DSPLNK utility. Each .ASM module above must be assembled first.

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

5.8.3 DSP96002 Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of DSP96002.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to DSP96002 source code, keeping in mind the necessary links between Hypersignal and the DSP96002 (see discussion in DSP96002 Source Code Interface Host PC Communication, above, and M96DEF.ASM module, which is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .CLN (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (entering an ".ASM" extension after "filename" is optional). This file executes a

```
asm96000 -B -L %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the M96LINK.CTL control file (use an ASCII text-editor to update the M96LINK.CTL file).

3) Re-link all DSP96002 modules using the Motorola DSPLNK linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .LNK files. LNK.BAT executes a

```
DSPLNK -BDSP96002.OUT -
MM96LINK.MAP -FM96LINK.CTL
```

statement, where "M96LINK.CTL" is an ASCII linker-control file that specifies all .CLN files to be linked by the DSPLNK.EXE program. Note that re-locatable code is positioned in internal PRAM between shared and local variables starting at 100H and software initialization code (M96BEG) starting at 400H (start of external PRAM), and afer M96BEG in external PRAM. This leaves significant space in internal PRAM for the linker to place interrupt service routines. You should not change absolute start of M96BEG at 400H unless you have a specific internal PRAM memory mapping strategy in mind; and you should not change the 100H address of shared variables for any reason, ever (as the host PC expects them there). All code has been tested using version 4.1 of ASM96000 and DSPLNK.

4) Copy the resulting "DSP96002.OUT" file to directory with Hypersignal, re-invoke the Hypersignal software, and run the appropriate function(s). Hypersignal reads the "COFF" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information:

DSP96002 Source Code Interface Modification Notes

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

DSP96002 Source Code Interface Modification Notes

1) If you need to change default buffer addresses, please consult the comments and suggestions in the M96DEF.ASM file.

2) The LNK.BAT file produces a "M96LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series

Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

3) The local variable "SEG" is used to store the current code segment page (upper 8 bits of linear address), and varies from board to board. Since virtually all DSP96002 program control instructions are not relocatable (instead using absolute addresses in the instruction sequence), all DSP96002 code must be dynamically relocated at run-time. After booting from internal PRAM, the DSP96002 code can then discover and set for future reference its code segment by examining instructions located in external PRAM.

4) In order to achieve the fastest possible means of transferring large amounts of continuous data in real-time between the host PC and the DSP96002, the Hypersignal-Macro implementation makes extensive use of the following combinations of transfer types, depending on the board in use:

a. onchip DMA between dual-port memory and other DSP96002 memory

b. onchip DMA between the host-interface and other DSP96002 memory

c. host-interface command-driven transfers

d. OnCE port driven transfers

It may be fair to say this is the most complex DSP chip interface supported by Hypersignal-Macro, of over 8 different DSP chips supported. It is definitely accurate

to say that this is one of the most interesting and fascinating.

5) The LSI DSP96002 board uses IRQB for analog sample interrupts, and IRQC to initiate on-chip DMA controller data transfers. IRQA is unused. The Ariel DSP-96 board uses IRQA for analog sample interrupts, and IRQB to initiate on-chip DMA controller data transfers. IRQC is left available for use with the "heartbeat" clock interrupt on the DSP-96.

6) In no case does the Hypersignal-Macro DSP96002 source code use a dedicated register set for its host-interface definition. All registers used are saved temporarily on the stack, allowing use of a 'C' or other high-level compiler.

7) If you need to use an external sampling clock, follow the basic guidelines below:

Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.

Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared

variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the

current menu (SAMPLING FREQ field).

Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

Motorola DSP96002 Source Code Interface for Hypersignal

DSP96002 Source Code Interface Interrupt Service Routines

DSP96002 Source Code Interface Host PC Communication

DSP96002 Source Code Interface Real-Time Processing Structure

DSP96002 Source Code Interface File List for Hypersignal

DSP96002 Source Code Modification Process for Hypersignal

5.9 AT&T WE DSP3210 Source Code Interface

Main Topics:

Related Topics:

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface Modification Notes

5.9.1 AT&T WE DSP3210 Source Code Interface for Hypersignal

The Hypersignal DSP3210 DSP Source Code Interface distribution diskette contains all AT&T WE DSP3210 source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic DSP3210 source code includes interrupt service routines for the Ariel DSP-3210 board and CAC QUANTUMdsp board.

As is, the DSP3210 source code is written in modular form, and supports the basic "modes" of operation below:

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the DSP3210 boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.9.1 above, AT&T WE DSP3210 Source Code Interface for Hypersignal) that the routine supports, for example "ISR0" for operating mode zero. Inside each interrupt service routine there are different entry points that contain hardware-specific

input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization ("itable" and PCW register) in RESET.S and W32BEG.S to locate the correct interrupt service routine for a particular function.

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

DSP3210 Source Code Interface Host PC Communication

If the user wishes to alter or add code functionality, or improve the functionality and/or performance of existing code, he/she must keep in mind the necessary intercommunication variables between Hypersignal software and DSP3210 code. The two-way communication between the host-PC and DSP3210 device set up in the Hypersignal software must not be changed, as the software expects to read/write certain memory locations. These memory locations

reside at locations "intercom" and "intermod" and are clearly commented inside W32DEF. *Do not insert any code of any type that would change the current addresses of these variables*. If necessary, you may add further communication variables for your own high-level code. The values of ANABFK, WNDBFK, RIBUFK, and MPBUFK (time data buffer, FFT window data buffer, real/imaginary data buffers, & magnitude/phase data buffers, respectively) can be changed if it becomes necessary to alter memory allocation in your DSP3210 code and communicate these changes to the host PC). The DSP3210 source code has been well-documented to aid the user in these areas.

For all DSP3210 boards, you should avoid using the area between 100H and 300H, which is used by Hypersignal to store host-DSP interface variables and structures (100H-1FFH) and local DSP variables (200H-3FFH).

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

DSP3210 Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground.

Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5", for a good example of a real-time, interrupt-driven structure.

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

5.9.2 WE DSP3210 Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal DSP3210 DSP Source Code Interface distribution diskette:

	reset vector, wait-state, and "page register" initialization
	variable & constant definition module
	initialization, including ISR vectors
	continuous A/D ISR for all board
	continuous D/A ISR for all

boards

Digital Oscillscope ISR for all boards

Spectrum Analyzer ISR for all boards

Stimulus & response ISR for all boards

Stimulus & response ISR for boards

analog I/O output ISR for Ariel DSP-3210 only

foreground A/D to disk processing (mode=0)

foreground D/A
from disk
processing
(mode=1)

foreground
Digital
Oscilloscope
processing
(mode=2)

FFT file-based
acceleration
processing
(mode=3)

iFFT file-based
acceleration
processing
(mode=4)

foreground
Spectrum
Analyzer
process
(mode=5)

foreground stim & response process (mode=6)

convolution/correlation file-based acceleration (mode=7)

arctangent routine

convolution/correlation routine

divide routine

complex FFT/iFFT routine of variable length N (N=2^{m}**m)

post-FFT processing (mag, phase, etc.)

filter control routine for real-time filtering

FIR filter routine

IIR (cascade sections, modified direct form) filter routine

IIR (cascade sections, balanced pole-zero structure) filter routine

log (base 10) approximation

routine

find maximum-in-buffer routine

find minimum-in-buffer routine

magnitude routine (complex input)

4-quadrant phase routine (–π to +π; –pi to +pi, complex input)

post-FFT processing for Spectrum Analyzer function, including mag,

phase, power-spectra, log, scaling, and data manipulation

squareroot routine

apply pre-defined window (used with FFTs)

complex-ratio routine used for transfer function

batch file used to assemble any source module: type "ASM filename" (no extension needed)

batch file used to link all source modules: type "LNK"

link control file

(command file for linker)

The distribution diskette does not initially contain all .O files necessary to produce a complete baseline, default executable code module when the W32LINK.CTL control file (above) is invoked with the AT&T d32ld utility. Each .S module above must be assembled first.

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

5.9.3 WE DSP3210 Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of DSP3210.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to DSP3210 source code, keeping in mind the necessary shared variable links between Hypersignal and the DSP3210 (see discussion in DSP3210 Source Code Interface Host PC Communication, above, and W32DEF.S module, which is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .O (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (*do not enter a ".S" extension after "filename"*). This file executes a

```
d32as -b l -lo %1 %1.s
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the W32LINK.CTL control file (use an ASCII text-editor to update the W32LINK.CTL file).

3) Re-link all DSP3210 modules using the AT&T DL3D linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .O files. LNK.BAT executes the following statements:

```
D32LD -abs -map -o
DSP3210.OUT W32LINK.CTL >
W32LINK.MAP
@COPY W32LINK.MAP+EOF
W32LINK.MAP
@D32NM -f DSP3210.OUT >> TEMP
@COPY W32LINK.MAP+TEMP
W32LINK.MAP
@DEL TEMP
```

Additional Information:

DSP3210 Source Code Interface Modification Notes

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

DSP3210 Source Code Interface Modification Notes

1) Hypersignal DSP3210 DSP source code uses r18 as a system stack pointer for both foreground and interrupt processing code. When storing CAU registers on the "stack", use the following sequence:

```
*r18++ = rN1
*r18++ = rN2
          -
          -
          -
*r18++ = rNN
```

When restoring CAU registers from the stack, use the following sequence:

```
rN1 = r18 – 4
rNN = *rN1--
          -
          -
          -
rN2 = *rN1--
rN1 = *rN1
r18 = r18 – K                /* K = 4 x (number of registers stored) */
```

There are three notable things about this process. First, the stack is always expected to increment and decrement in 4-byte chunks (always save/restore extended registers). Second, the stack is always expected to point to the *next available (free) location*. Third, when restoring, you can't simply do an r18e = r18 – 4 instruction and begin popping rXX = *r18-- instructions, because an interrupt

might occur sometime after the r18 – 4 instruction and sometime before you finish popping; the interrupt routine, if it also used r18 as a stack pointer (which all types of routines should be able to do of a legitimate stack pointer), might then clobber a stored register (or more than one) that hadn't been popped yet.

2) Hypersignal DSP3210 source code is built with the "ASM" and "LNK" batch files, which invoke the AT&T "d32as" and "d32ld" assembler and linker utilities, respectively. By examining the ASM and LNK batch files, you should be able to construct an equivalent method of building the code using the AT&T "make" utility instead; however, producing a memory map file will be difficult. For the DSP Source Code Interface, obscure DOS commands have been used to create a map file in the LNK.BAT file, since the AT&T utilities will only print map output to the screen. The map file produced has a filename of "W32LINK.MAP"; you can use this file to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory,

problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

3) In addition to setting them directly in the DSP3210 source code, IOC register settings can be controlled from the high-level software in order to accommodate non-standard analog I/O connected to DSP3210 serial ports. Inside the Hypersignal-Macro configuration file there is a 32-bit integer IOC value for every DSP3210 board currently supported by Hypersignal-Macro--see the "CONFIG.DOC" file on Hypersignal distribution diskette #2 for a list of the sequence of variables in the configuration file. Depending on the board currently in use, one of these values is sent down to the DSP3210 board whenever a Hypersignal-Macro real-time function requiring analog I/O is invoked. The corresponding DSP source code variable is "IOC_reg", which is normally used directly to set the IOC register (unless changed by the user in the DSP source code).

4) If you need to use an external sampling clock, follow the basic guidelines below:

Set any jumpers required on the

board itself. See the jumper list and/or hardware manufacturer documentation.

Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source

code as appropriate.

If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

AT&T WE DSP3210 Source Code Interface for Hypersignal

DSP3210 Source Code Interface Interrupt Service Routines

DSP3210 Source Code Interface Host PC Communication

DSP3210 Source Code Interface Real-Time Processing Structure

DSP3210 Source Code Interface File List for Hypersignal

DSP3210 Source Code Modification Process for Hypersignal

5.10 Texas Instruments TMS320C4x Source Code Interface

Main Topics:

Related Topics

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface Modification Notes

5.10.1 Texas Instruments TMS320C4x Source Code Interface for Hypersignal

The Hypersignal TMS320C4x DSP Source Code Interface distribution diskette contains all Texas Instruments TMS320C4x source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic TMS320C4x source code includes interrupt service routines for the Ariel DSP-C40 board.

As is, the TMS320C4x source code includes both library routines and also real-time interrupt driven "modes" of operation that use foreground processing and interrupt service routine processing to implement standard Hypersignal functions. These functions have been written in modular form, and support the basic "modes" of operation below:

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the TMS320C4x boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.10.1 above, Texas Instruments TMS320C4x Source Code Interface for Hypersignal) that they support, for example "ISR0". Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine the interrupt vector and mask initialization in the initial setup code in "T40BEG", depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

TMS320C4x Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she must keep in mind the necessary links between Hypersignal software and the TMS320C4x code. The two-way communication between the PC and TMS320C4x processor established in the Hypersignal software must not be changed, as the high-level software expects to read/write certain memory locations. These memory locations are initialized by the high-level software before any code executes on the DSP board (some locations are actually read by the high-level software as well, allowing you to change such "constants" as buffer base addresses). *Do not insert any code of any type before these addresses*. The TMS320C4x source code has been documented to aid the user in these areas.

For all TMS320C4x boards, you should avoid using the program memory areas between 0CH and 0C0H, which are reserved by Texas Instruments, between 0C0H and 400H, which is often used for debuggers and monitors, between

400H and 47FH, which is used by Hypersignal to store host-DSP interface variables and structures, and between 480H and 4FFH, which is used by Hypersignal TMS320C4x DSP code to store local variables.

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

TMS320C4x Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. The dual-buffer communication between the host and DSP is fully synchronized using "interlock"

buffer flags. Use mode 5 source modules ("MODUL5" and routines it calls) and "xxx_ISR5" for one example of a real-time, interrupt-driven structure.

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

5.10.2 TMS320C4x Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal TMS320C4x DSP Source Code Interface distribution diskette:

variable & constant definition module	
	initialization, including ISR

vectors

continuous A/D ISR for all boards

continuous D/A ISR for all boards

Digital Scope ISR for all boards

Spectrum Analyzer ISR for all boards

stimulus &

response ISR
for all boards

foreground
A/D to disk
processing
(mode=0)

foreground
D/A from disk
processing
(mode=1)

foreground
Digital
Oscilloscope
processing
(mode=2)

FFT file-based
acceleration
processing
(mode=3)

iFFT file-based acceleration processing (mode=4)

foreground Spectrum Analyzer process (mode=5)

foreground stimulus & response process (mode=6)

convolution/correlation file-based acceleration (mode=7)

arctangent routine

convolution/correlation routine

floating-point divide routine

complex FFT/iFFT routine of variable length N ($N=2^{m}$ **m)

post-FFT

processing (mag, phase, etc.)

log approximation routine

magnitude routine (complex input)

find maximum-in-buffer routine

find minimum-in-buffer routine

4-quadrant phase routine (−π to +π; −pi to +pi,

complex input)

sine value lookup-table for FFT use

post-FFT processing for Spectrum Analyzer function, including mag, phase, power-spectra, log, scaling, and data manipulation

squareroot routine

apply pre-defined window (used with FFTs)

filter control routine for real-time filtering

FIR filter routine

IIR (cascade sections, modified direct form) filter routine

IIR (cascade sections, balanced pole-zero structure) filter routine

complex-ratio routine used for transfer function

	batch file used to assemble any source module: type "ASM filename" (no extension needed)
	batch file used to link all source modules: type "LNK"
	link control file (command file for linker)

The distribution diskette does not initially contain all .OBJ files necessary to produce a complete baseline, default executable code module when the T40LINK.CMD control file (above) is invoked with the Texas Instruments LNK40 utility. Each .ASM module above must be assembled first.

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

5.10.3 TMS320C4x Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of TMS32040.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to TMS320C4x source code, keeping in mind the necessary shared variables between Hypersignal and the TMS320C4x (the T40DEF.ASM module is well-documented in this area).

2) Assemble all modules changed and any new modules created to ensure a new .OBJ (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that you may use in the form of:

```
ASM filename <car ret>
```

to assemble source code modules (entering an ".ASM" extension after "filename" is optional). This file executes a

```
asm30 -l -v40 %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the T40LINK.CMD control file (use an ASCII text-editor to update the T40LINK.CMD file).

3) Re-link all TMS320C4x modules using the Texas Instruments LNK30 linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ files. LNK.BAT executes a

```
lnk30 -v40 T40LINK.CMD
```

statement, where "T40LINK.CMD" is an ASCII linker-control file that specifies all .OBJ files to be linked by the LNK30.EXE program. Note that T40LINK.CMD places shared and local variables at 400H and 480H, respectively. All code has been tested using version 4.5 of the Texas Instruments TMS320C3x/4x development tools.

4) Copy the resulting "TMS32040.OUT" file to the directory with Hypersignal software, run Hypersignal, and invoke the appropriate functions. Hypersignal reads the "COFF" file

format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information:

TMS320C4x Source Code Interface Modification Notes

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

TMS320C4x Source Code Interface Modification Notes

1) Avoid using the default ".TEXT", ".DATA", or ".BSS" sections in any new code you might add. For some reason, the Texas Instruments linker wants to put these sections in front of all named sections, unless you specifically tell it otherwise in the .CMD file. If one of these sections loads in front of the "INIT" section, which is supposed to load at zero because it contains the reset vector, your code is dead.

2) *Do not declare new variables before or in the "HOST_COMM" section (file T40DEF.ASM).* These are the host-DSP

intercommunication memory areas that must remain unchanged to allow normal Hypersignal operation.

3) Note that the start of executable code used by Hypersignal is at 500H--if you declare enough variables in section "LOCAL" (file T40DEF.ASM) or immediately after it, you could overrun the start address. If necessary put the variables somewhere else or change the address of "INIT" (both in T40BEG.ASM and T40LINK.CMD). Also note that analog and acceleration processing data buffers start at address 2000H. If you add enough code, you could overrun this address. If so, change the buffer address value "ANABFK" only--Hypersignal will base all other buffers on this value. *If you need to move any buffer addresses, please see the T40DEF.ASM file--it contains memory maps and numerous suggestions on changing default addresses*.

4) The LNK.BAT file produces a "T40LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the

available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter 3 for hardware-specific memory amounts.

5) If you need to use an external sampling clock, follow the basic guidelines below:

Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.

Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will

be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". This value is in Hz (Hertz) and is initialized by the high-level software, based on what is entered in the current menu (SAMPLING FREQ field).

Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling

clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

Texas Instruments TMS320C4x Source Code Interface for Hypersignal

TMS320C4x Source Code Interface Interrupt Service Routines

TMS320C4x Source Code Interface Host PC Communication

TMS320C4x Source Code Interface Real-Time Processing Structure

TMS320C4x Source Code Interface File List for Hypersignal

TMS320C4x Source Code Modification Process for Hypersignal

5.11 Texas Instruments TMS320C5x Source Code Interface

Main Topics:

Related Topics

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface Modification Notes

5.11.1 Texas Instruments TMS320C5x Source Code Interface for Hypersignal

The Hypersignal TMS320C5x DSP Source Code Interface distribution diskette contains all TMS320C5x source code, including linker control files and support utilities. It is highly recommended that you make a backup copy of this disk before making any changes.

The basic TMS320C5x source code includes analog I/O interrupt service routines for the Innovative Integration PC50 board.

As is, the TMS320C5x source code is written in modular form, and supports the basic "modes" of operation below:

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

There are various interrupt service routines that respond to interrupts on the TMS320C5x boards and generally match up with the foreground processing modules above. Interrupt service routine module names consist of "ISRn", where n indicates the operating mode (see section 5.11.1 above, Texas Instruments TMS320C5x Source Code Interface for Hypersignal) that they support, for example "ISR0". Inside each interrupt service routine there are different entry points that contain hardware-specific input/output code for individual boards. These hardware-specific sections are kept as small as possible, so that the great majority of the routine is common for all boards. Examine

the interrupt vector and mask initialization in the initial setup code in "T50BEG", depending on board type (see host-shared variable "BOARD"), to locate the correct interrupt service routine for a particular function.

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

TMS320C5x Source Code Interface Host PC Communication

If the user wishes to alter the performance of the code, he/she must keep in mind the necessary links between Hypersignal software and the TMS320C5x code. The two-way communication between the PC and TMS320C5x established in the Hypersignal software must not be changed, as the high-level host software expects to read/write certain memory locations. These memory locations are initialized by table reads from program memory inside module T50BEG, and are suffixed by 'ADD', as in MODADD, TRGADD, etc. *Do not insert any*

code of any type before these addresses that would cause them to move. The TMS320C5x source code has been documented to aid the user in these areas.

For all TMS320C5x boards, you should avoid writing to program memory between 26H and 3FFH, which is typically left open for TMS320C5x debuggers, and 42DH and 4FFH, which is used by Hypersignal to store host-DSP communication variables and structures. Also, note that if Hypersignal TMS320C5x code is allowed to run, it stores local variables in onchip data memory locations 0-127, location 2FF, and 896 through 1023 (filter coefficients and delay elements).

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

TMS320C5x Source Code Interface Real-Time Processing Structure

Currently, operating mode 5 (Spectrum Analyzer) is the best example of full implementation of

foreground processing with complete context-save background interrupts. The Spectrum Analyzer source code represents a totally real-time system, with samples buffered in a dual-buffer scheme in response to background "sample ready" interrupts, and buffers, when full, sent to FFT and post-FFT processing in the foreground. Use mode 5 source modules ("MODUL5" and modules it calls), and "ISR5" for one example of a real-time, interrupt-driven structure.

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

5.11.2 TMS320C5x Source Code Interface File List for Hypersignal

Source code and related files included on the Hypersignal TMS320C5x DSP Source Code Interface distribution diskette:

variable & constant definition

beginning vectors, including ISR vectors, and including initialization, including PC-written variables

continuous A/D ISR for all board

continuous D/A ISR for all boards

Digital Scope ISR for all boards.

Spectrum
Analyzer ISR
for all boards.

Stimulus &
Response ISR
for all boards.

foreground
A/D-to-disk
processing
(mode=0)

foreground
D/A-from-disk
processing
(mode=1)

foreground
Digital
Oscilloscope
processing
(mode=2)

FFT file-based acceleration processing (mode=3)

iFFT file-based acceleration processing (mode=4)

foreground Spectrum Analyzer process (mode=5)

foreground stimulus & response (mode=6)

convolution/correlation file-based acceleration processing (mode=7)	
arctangent routine	
convolution/correlation routine	
fractional and integer divide routines	

exponentiation (base-2) routine (needed for fixed-point transfer function calculation)

complex FFT/iFFT routine of variable length N (N=2**m)

post-FFT processing (mag, phase, scaling)

invokes correct filter for current channel

FIR filter routine

IIR filter cascade struc 2 (modified direct-form)

IIR filter cascade struc 3 (balanced pole-zero)

log2 approximation routine

magnitude routine

find maximum routine

find minimum routine

move data

4-quadrant phase routine (−π to +π; −pi to +pi, complex input)

sine value lookup-table for FFT use

post-FFT processing for Spectrum Analyzer function, including mag,

phase, power-spectra, log, scaling, and data manipulation

squareroot routine

apply pre-defined window (used with FFTs)

complex-ratio routine used for transfer function

batch file used to assemble any source module: type "ASM filename" (no extension needed)

batch file used to link all

source modules: type "LNK"

link control file (batch file for linker)

The distribution disk does not initially contain all .OBJ files necessary to produce a complete baseline, default executable code module when the T50LINK.CMD control file (above) is invoked with the Texas Instruments "dsplnk" utility. Each .ASM module above must be assembled first.

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

5.11.3 TMS320C5x Source Code Modification Process for Hypersignal

NOTE: Be sure to save a backup copy of TMS32025.OUT before running the "LNK" batch file in case your new code doesn't work.

Below are the steps taken to modify, add, and integrate new source code:

1) Make changes to TMS320C5x source code, keeping in mind the necessary shared variables between Hypersignal and the TMS320C5x. The file T50DEF.ASM is well-documented in this area.

2) Assemble all modules changed and any new modules created (using Texas Instruments "COFF" assembler) to ensure that a new .OBJ (object) file is generated for each altered module. For convenience, an "ASM.BAT" batch file has been provided that can be invoked as:

```
ASM filename <car ret>
```

to assemble source code modules (entering the ".ASM" extension after "filename" is optional). This file executes a

```
dspa -l -v50 -w %1
```

statement, where %1 is the filename.

If a new module has been created, be sure to add it to the T50LINK.CMD control file (use an ASCII text-editor to update the T50LINK.CMD file).

3) Re-link all modules using Texas Instruments TMS320C5x "dsplnk" linker. A "LNK.BAT" batch file has been provided that you may use by invoking:

```
LNK <car ret>
```

to link all necessary .OBJ files. LNK.BAT executes a

```
dsplnk T50LINK.CMD
```

statement, where "T50LINK.CMD" is an ASCII linker-command file that specifies all .OBJ files to be linked by the dsplnk.exe program, as well as specific information about memory addresses for some sections. Note that all executable code that is re-locatable starts at 400H. All code has been tested using version 6.40 of "dspa.exe" and "dsplnk.exe".

4) Copy the resulting "TMS32050.OUT" file to directory with Hypersignal software, run Hypersignal, and invoke the appropriate functions. Hypersignal reads the "COFF" file format directly, downloads all code and data, and begins execution at the reset vector (location zero).

Additional Information:

TMS320C5x Source Code Interface Modification Notes

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

TMS320C5x Source Code Interface Modification Notes

1) If you need to change default buffer addresses, please consult the comments and suggestions in the T50DEF.ASM file.

2) The LNK.BAT file produces a "T50LINK.MAP" map file which you can use to check for overlaps and current locations of variables, routines, and code sections. The map file also shows the starting address and length of each module included in the current executable output file. As you add significant amounts of code, you should pay extra attention to the length of the executable file as compared the amount of program memory available. If you exceed the available program memory, problems can occur that seem to result from other causes, and which can lead you to spend unnecessary time searching for the real cause. See the "Onboard Memory Usage" sections for individual boards in Hypersignal-Macro Series Hardware Reference Guide chapter

3 for hardware-specific memory amounts.

3) The T50LINK.CMD file creates an uninitialized "hole" from 26H through 3FFH that can be used for monitor and debugger programs. Because this section is uninitialized, the Hypersignal COFF loader will not write anything to this area.

4) If you need to use an external sampling clock, follow the basic guidelines below:

Set any jumpers required on the board itself. See the jumper list and/or hardware manufacturer documentation.

Enter '-EXT' after the board designator in the Hypersignal System Configuration menu. This causes the high-level software to set the value of the shared variable "FS_MODE" as appropriate for external clock operation. If no setting is possible due to the design of the board, then the value of FS_MODE will

be zero. If the high-level software does not set the value correctly, or you introduce new external sampling circuitry to the hardware, we suggest that you continue to use the FS_MODE variable to control the hardware, by directly changing the DSP source code as appropriate.

If you need to know the actual sampling rate value, this is contained in the shared variable "FS_VALUE". The value is in Hz (Hertz); note that for TMS320C5x boards, which use 16-bit wide memory accesses, the maximum value that can be represented is limited to 65535. This value is initialized by the high-level software, based on what is

entered in the current menu (SAMPLING FREQ field).

Enter SAMPLING FREQ field values in all high-level menus to match the actual external sampling clock rate, so that time and frequency unit readouts in all real-time and post-acquisition displays are accurate.

Related Topics

Texas Instruments TMS320C5x Source Code Interface for Hypersignal

TMS320C5x Source Code Interface Interrupt Service Routines

TMS320C5x Source Code Interface Host PC Communication

TMS320C5x Source Code Interface Real-Time Processing Structure

TMS320C5x Source Code Interface File List for Hypersignal

TMS320C5x Source Code Modification Process for Hypersignal

6 Using DSPower®-Block Diagram Software

Overview

General Capabilities

Depending on the DSP Engine(s) being used (see below), general capabilities in the DSPower environment include:

1. construct an automated test and/or measurement sequence, including DSP/Analog hardware interface, that runs on the host machine
2. simulate a DSP/math algorithm or system on the host machine
3. access interactive instruments to analyze and measure physical signals connected to supported DSP/Analog hardware
4. execute a DSP/math algorithm or system in real-time, using supported DSP/Analog hardware containing a supported DSP chip
5. access visualization displays to analyze and measure waveform files, arrays, data sets, matrices, etc.
6. use block-diagram interface and source code generation and manipulation capability as a CASE tool to allow design, development, and maintenance of source code for DSP/math systems

7. accelerate a DSP/math algorithm with supported DSP hardware

Specific Capabilities

Depending on the DSP Engine(s) being used (see below), specific capabilities accessible in the DSPower environment include:

1. DSP / math functions
2. visualization displays, including x-y plot, waterfall, contour, pole-zero, linear and log magnitude, unwrapped phase, group delay, and more(depending on the DSP Engine, multiple displays can be shown at once, and multitasking is possible)
3. instrumentation, including Spectrum Analyzer, Digital Oscilloscope, Stimulus & Response measurement, and continuous function generation(depending on the DSP Engine, instruments can be run simultaneously with other tasks)
4. data acquisition(continuous waveform file store and generate to / from disk, digital tape recorder)
5. waveform editing
6. IIR / FIR filter design
7. support for 55+ types of DSP / Analog, acquisition, and multimedia hardware, and more than 8 popular DSP chips
8. import of waveform, array, and matrix data in various waveform file formats
9. export of waveform, array, and matrix data to ASCII files suitable for use by other applications

Main Topics

Getting Started

Creating Block Diagrams

Adding User Defined Blocks

Interactive Operation

Debugging Diagrams

6.1 Getting Started

After installing the software, a new program group called "Signalogic DSP Software" will be created in the Windows Program Manger (it may already be there if you have installed another Signalogic DSP software package, such as DSPower-HWLib DLL or Hypersignal-Macro). Double-click on the “DSPower-Block Diagram” icon to start.

For full installation details, refer to System Guide section 3.2, Installing Signalogic DSP Software.

Helpful Things to Know When Getting Started

1) Almost all buttons and selections in DSPower-Block Diagram have “fly-over” context-sensitive help. Leave your cursor still over a button and a fly-over help box should pop up with a brief description. Often times the fly-over help box has a small help button inside, which you can click to bring up more detailed information.

2) DSPower-Block Diagram requires at least oneof three (3) possible “DSP Engines” in order to operate: Hypersignal (-Macro, -Acoustic, or -Macro NRT), MATLAB, or a supported DSP board combined with the DSPower C Code Generator. See System Guide sections 1.1, Relationship of Signalogic DSP Software

Components and 1.2, Summary of Signalogic DSP Software Components.

Although the directory paths of installed DSP Engines should already be set during installation of DSPower software, if you have any problems running blocks, like error messages saying “...engine not found...” or similar, you should check the DSP Engines item under the Configuration main menu heading.

3) The easiest way to add a block in DSPower is to point to the desired location in a diagram, and double-click. The block type currently highlighted in the Block Selector (see section 6.1.3, below) will appear as a symbol, with a text label describing the block. The second easiest method is to drag a block type to the desired location,and drop. This method has the added convenience of causing the block setup parameters to appear also.

4) When blocks execute in DSPower-Block Diagram, there is a “DSP Engine Status” box that should pop up to tell you what is going on. This box contains a running report of the status of executing blocks and diagrams (you can also scroll back to review previous events for debugging or other purposes). This status box is vey important; without it you may not realize when an error condition has occurred, or when a block or diagram has completed execution. If the DSP Engine status box is not

visible, select the Diagram item under the Options main menu heading and make sure either of the selections “pop up with new info” or “always visible” are set.

5) There are two helpful floating dialogs that can be activated (and put away) as needed: the Block Selector, and the Waveform Locator. These two dialogs are activated by the “find” buttons on the main toolbar (the ones with pictures of binoculars).

6) Before a block that uses DSP/analog hardware in some way will operate correctly, you must select the DSP/Analog Hardware Configuration toolbar button or main menu item (under Configuration main menu heading). Inside this dialog box, you must also click on Set Hardware Variables, and make current selections (typically, for a system with only one DSP/analog board installed, you can use “Set All” inside this dialog box).

Examples of blocks that require hardware configuration before operating correctly include all instrument blocks, function blocks that use a DSP accelerator, and display blocks capable of D/A output. Also, some main toolbar functions, like the DSPower Debugger, also require hardware setup.

7) Maximizing performance of the Windows 3.1x and Windows 95 operating systems can be a complex process, with many

settings to consider, and the applications which run under them are no different. There are many system-wide and local settings which affect performance of DSPower-Block Diagram, andwhich should at least be given the “once over”. For more information, see System Guide sections 1.4, System Considerations and 1.4.2, Performance Considerations.

8) Although the location of waveform files should already be set during installation, if you get error messages like “...unable to find waveform file...” then you should check the Directories item under the main menu heading Configuration.

6.1.1 Installation and Checkout

Refer to System Guide sections 3.2, Installing Signalogic DSP Software and 3.2.4, Configuration and Checkout of DSPower-Block Diagram Software.

Related System Guide topics:

Installation Overview

6.1.2 Online Help in DSPower

DSPower offers extensive online documentation in the form of three help files (see listing below)

Help File	Topic
supguide.hlp	The System Guide, overall reference to system configuration, installation, documentation conventions, troubleshooting etc.
useguide.hlp	The User's Guide, help on how to use Signalogic DSP Software.
refguide.hlp	The Signalogic DSP Software Reference Guide.

Also, navigating through help is made very simple. See topics below for more information:

Using Help in DSPower

Search++ Option

This option allows search for topics across multiple help files. This option presents users with a menu of the available help files and enhances the search capability. DSPower is installed with one default setting. For information on customizing this option see Help Options.

To use this option:

- Open help from the main Help menu
- Click on the Search++ button and chose any file from the available list

See Also:

Dialog Boxes

Context-Sensitive Help

Dialog Boxes

Help buttons are made available in all dialog boxes in DSPower. Clicking on these buttons presents help on the options available from the entire dialog box. Context-sensitive help is also available on the individual options in the dialog box.

See Also:

Search++

Context-Sensitive Help

Context-Sensitive Help

Context-sensitive help is available for most topics in DSPower in the form of button help. The little help icon on the button help window opens the topic in the main help file.

See Also:

Search++

Dialog Boxes

6.1.3 Block Selector

The DSPower block selector lists data types and source code types across the top, and block types down the side. Within this matrix, depending on which data, source and block type is selected, are a list of available blocks. From this list, you chose an item. To change the list, you simply change the "matrix" elements across the top side.

Block Selector options:

Data Type

This option allows choice of type of data to be used with software. The available data types are DSP(vector data), IMG(image) or XY.

DSP Data Type

IMG Data Type

XY Data Type

Other Options:

Source Code Type

Block Type

Block List

Find Block

Block Selector Toolbar

Source Code Type

This options allows choice of the type of generated source. Depending on the DSP Engine used, chose .MAC for Hypersignal Macro / Acoustic, .MAT for MATLAB and .c/.cpp for C Code Generation.

An improvement that could be considered in future is to make this option independent of the DSP Engine being used.

.mac Souce Code Type

.m Source Code Type

.c/cpp Source Code Type

Other Options:

Data Type

Block Type

Block List

Find Block

Block Selector Toolbar

Block Type

Available Block types are listed here. Block types include control constructs (if-then, while, for), functions, displays, instruments, filter designs and implementations, utilities, and code blocks. (a "raw" piece of source code, with which nobody knows what to do except insert it into the algorithm). The required block can be selected from this list.

Code Blocks

lsdjfsdjfl

Control Flow Blocks

Filter Blocks

Function Blocks

Instrument Blocks

Monitor/Display Blocks

Utility Blocks

Other Options:

Source Code Type

Data Type

Block List

Find Block

Block Selector Toolbar

Block List

Other Options:

Source Code Type

Data Type

Block Type

Find Block

Block Selector Toolbar

Find Block

Other Options:

Source Code Type

Data Type

Block Type

Block List

Block Selector Toolbar

Block Selector Toolbar

Other Options:

Source Code Type

Data Type

Block Type

Block List

Find Block

6.1.4 Worksheet

The worksheet is presented when you first enter DSPower. Within the worksheet are two important controls, implemented as windows so you can position them as needed, or remove them temporarily. The first one is the "Block Selector." The basic user interface is like most other Windows programs: point the mouse and double-click. Double clicking on a list item causes the block to appear at the lower left of the worksheet. Pointing the mouse and then double-clicking causes the block to appear where you want it. The second control is the "toolbar.".

See Worksheet Options for more information about customization of the worksheet.

Worksheet Toolbar

The toolbar in the DSPower worksheet contains several frequently used operations in icon form. The icons available are listed below:

Run Block Diagram

Stop Block Diagram

Procedurize block

Waveform Locator

Show / Hide MATLAB command window

6.1.5. Waveform Locator

[@k Locating Waveform Files]

This utility provides quick way to locate waveform files in the waveform directory. It also allows drag & drop operation. For Further information. see Dragging Waveform Files.

Available Options:

Attributes

Selecting this option lists other waveform attributes like number of channels, whether complex or not, file data type etc.

See Also:

Headitor

Waveform Search Criteria

Headitor

This utility is a short name for "Header Editor" and allows parameter-by-parameter inspection and modification of waveform file headers The Headitor uses the filename field entry and displays all header parameters in the respective fields. If no filename is entered, then the Headitor will display null entries for all parameters until a filename is entered. If no extension is entered, a .tim extension is assumed.

See Also:

Attributes

Waveform Search Criteria

Waveform Search Criteria

This section allows you to simplify your search by providing the following options:

Waveform path

Lists all specified type of waveform files in the waveform path the system is configured to. The default path is c:\waveforms. For more information see Waveform Files.

File Spec

Lists all waveform files with given specifications from the entry in the File Spec field.

Time Domain

Lists all Time Domain(.tim) files in the waveform path specified in the Waveform Files field in the Directory Configuration menu.

Freq Domain

Lists all Frequency domain(.frq) files in the waveform path specified in specified in the waveform files field in the directory configuration menu.

Wavelet Domain

Lists all Wavelet domain (.wlt) files in the waveform path specified in the waveform files field in the directory configuration menu.

Filter Designs

Lists all Filter Design (.ffd) files in the waveform path specified in the waveform files field in the directory configuration menu.

Filter Coeffs

Lists all Filter Coefficient (.fir) files in the given path specified in the waveform files field in the directory configuration menu.

See Also:

Attributes

Headitor

Dragging Waveform Files

Drag and Drop operation is supported in DSPower and can be used to load Waveform Files. Drag Drop can be done as follows:

1. Select the waveform(s) you would like to load.

2. Click the mouse button; the drag & drop icon appears. Drag the waveform and drop it on the block you would like to load it to. If the waveform is dropped on the worksheet, it is loaded onto the active block in the Block Selector.

3. The Load/Run Query is activated. Chose the appropriate option. See Diagram Display Options for more information about deactivating this query.

6.1.6 Style/Setup Locator

[@k Locating Setup/Style Files]

This utility provides a quick search method of all Setup/Style Files in the waveform directory. The default path is c:\waveforms.

Setup Search Criteria

This section allows you to simplify your search by providing the following options:

Waveform path

Lists all specified type of waveform files in the waveform path the system is configured to. The default path is c:\waveforms. For more information see Waveform Files

File Spec

Lists all waveform files with given specifications (entry in the File Spec field)

Setups

Lists all Waveform Setup(.set) files in the waveform path specified in the Waveform Files field in the Directory Configuration menu.

Styles

Lists all Waveform Style(.sty) files in the waveform path specified in specified in the waveform files field in the directory configuration menu.

Code Blocks

Lists all specified file types corresponding to Code Blocks in the waveform path specified in the waveform files field in the directory configuration menu.

Ctrl Constructs

Lists all specified file types corresponding to Control Constructs in the waveform path specified in the waveform files field in the directory configuration menu.

Displays

Lists all specified file types corresponding to Display Blocks in the waveform path specified in the waveform files field in the directory configuration menu.

Filters

Lists all specified file types corresponding to Filter Blocks in the waveform path specified in the waveform files field in the directory configuration menu.

Functions

Lists all specified file types corresponding to Function Blocks in the waveform path specified in the waveform files field in the directory configuration menu.

Instruments

Lists all specified file types corresponding to Instrument Blocks in the waveform path specified in the waveform files field in the directory configuration menu.

Utilities

Lists all specified file types corresponding to Utility Blocks in the waveform path specified in the waveform files field in the directory configuration menu.

6.2 Options in DSPower

6.2.1 Program Options

Program Display Options

Object size independent of display resolution

Save settings on exit

6.2.2 Worksheet Options

The following options allow for customization of the DSPower Worksheet. The available options are listed below.

Worksheet Display Options

Block Selector Visible

Selecting this option makes the Block Selector visible. This is selected by default in DSPower.

Block Selector always on top

Waveform locator visible

Minimize block selector after each selection

This option allows minimization of the block selector after a block is selected.

Waveform locator always on top

Toolbar Visible

Selecting this option makes the Worksheet Toolbar visible. This is selected by default in DSPower.

Satellite view enabled

This option enables satellite view.

Other Options

Background Color

External Program Windows

External Program Windows

Various DSP Engine Info Windows are made available in DSPower to

facilitate diagram operations and debugging.

Status

The status window presents status messages as the blocks are running. This window appears when a block/diagram is run, i.e. with any new information. This default condition can be overridden by selecting the required option.

Waveform Summary

The waveform summary window provides quick information about waveforms used in the diagram. This window appears by default whenever a block/diagram is run, i.e. with any new information. The default condition can be overridden by selecting the required option.

Waveform File Directory

The waveform file directory window presents a directory of available waveform files in the waveforms directory. This window appears with any new information it has. This default condition can be overridden by selecting the required option.

Other Options

Worksheet Display Options

Background Color

Background Color

This option allows selecting/changing of the background color for the DSPower Worksheet.

6.2.3 Diagram Options

The following options are made available in DSPower to allow customization of diagrams.

Diagram Display Options

The following options allow for the customization of the Diagram in DSPower. The available options are listed below.

Enable Symbol Display within Blocks

This option enables symbol display within blocks.

Snap to Grid

Snap to grid; Initially disabled

Enable connection avoidance for all objects

Enables connection

Ex Parameters Always Visible

Object size independent of Display resolution

Selecting this option allows for object sizes to be independent of resolution.

Text Labels Visible

Each block in DSPower has a text label which is usually the name of the block. These labels are visible by default. Turning of this option disables the text labels. If enabled, they can be configured to appear either above or below the blocks.

Drag-Dop Action Fixed

Align Blocks to Flow Stage Grid

Disp/Inst. file menu

Disp/Inst file menu

Disable Load/Run query

This option allows enabling/disabling the Load/Run query while Dragging Waveform Files.

Other Options

Diagram Flow Options
Background Color
Block Boundary
DSP Engine Info Windows

Diagram Flow Options

Each block in DSPower has two types of connectors: The Data Connector and the Control Connector. The data connector specifies inputs and outputs of the block. The control connections determine the order of execution of blocks in a diagram.

This option allows the choice of data connections, control connections or both (default).

Other Options

Diagram Display Options

Background Color

Block Boundary

DSP Engine Info Windows

Background Color

This option allows for selecting/changing of background color for any diagram.

Other Options

Diagram Display Options

Diagram Flow Options

Block Boundary

DSP Engine Info Windows

Block Boundary

This option specifies the Block Boundary.

Other Options

Diagram Display Settings

Diagram Flow Options

Background Color

DSP Engine Info Windows

DSP Engine Info Windows

Various DSP Engine Info Windows are made available in DSPower to

facilitate diagram operations and debugging.

Status

The status window presents status messages as the blocks are running. This window appears when a block/diagram is run, i.e. with any new information. This default condition can be overridden by selecting the required option.

Waveform Summary

The waveform summary window provides quick information about waveforms used in the diagram. This window appears by default whenever a block/diagram is run, i.e. with any new information. The default condition can be overridden by selecting the required option.

Waveform File Directory

The waveform file directory window presents a directory of available waveform files in the waveforms directory. This window appears with any new information it has. This default condition can be overridden by selecting the required option.

Other Options

Diagram Display Options

Diagram Flow Options

Background Color

Block Boundary

6.2.4 Display Options

Interactive Display Options

Disp/Inst File Menu

Enable Menus

Other Options

Overlaid Toolbar Options

Overlaid Toolbar Options

Percent Max Display Area

Autohide

Disable Overlay When Toolbars Can Fit

Other Options

Interactive Display Options

6.2.5 Help Options

Online help is very powerful in DSPower. The following options

are provided for customizing the help windows.

Online Help Settings

Fly-Over Help Enabled

Button Help Display Enabled

DSPower offers very powerful Context-Sensitive Help in the form of Button Help Display. This help can be either long or short form depending on the option chosen.

Help Windows always on Top

Selecting this option sets the "Always on Top" attribute for all help windows. The help window then stays above the Worksheet until it is closed.

Save Help Window Position Between Sessions

Selecting this option saves window position between sessions.

Allow Restore To Maximized

This option allows restore to be maximized.

Enable Navigator

Other Options

Help File List

Help File List

A list of all Help Files used with DSPower is listed in this list box.

Options to add/delete help files are also available.

Other Options

Online Help Settings

6.3 Configuring DSPower

Before you can start using DSPower, you will need to configure the following fields:

6.3.1 Directory Configuration

Setting the fields in this menu option tells the software where to look for Waveform files, Block Symbol files, system source template files, used defined template files, etc.

Directory Configuration Fields

Diagram Files

This field specifies the location of all diagram files used by the software and is initialized to c:\dspower\diagrams. If installation was not done with the defaults, the entry in this field has to be changed.

Other Configuration Fields:

Block History Files

Block Symbol (Bitmap) Files

DSP Chip Executable Files
System Source Templates
User Source Templates
Waveform Files

Waveform Files

This field specifies the location of all waveform files used by the software and is initialized to c:\waveforms. If installation was not done with the defaults, the entry in this field has to be changed.

Other Configuration Fields

Block History Files
Block Symbol (Bitmap) Files
Diagram Files
DSP Chip Executable Files
System Source Templates
User Source Templates

System Source Templates

This field specifies the location of source template files available in the system library and is initialized to c:\dspower\syslib. If installation was not done with the default settings, the entry in this field has to be changed.

Other Configuration Fields

Diagram Files

Block History Files
Block Symbol (Bitmap) Files
DSP Chip Executable Files
User Source Templates
Waveform Files

User Source Templates

This field specifies the location of any user defined source template files and is initialized to c:\dspower\userlib. If installation was not done with the default settings, the entry in this field has to be changed.

Other Configuration Fields

Diagram Files
Block History Files
Block Symbol (Bitmap) Files
DSP Chip Executable Files
System Source Templates
Waveform Files

Block Symbol (Bitmap) Files

This field specifies the location of bitmap files used by the software. If installation was not done with the default settings, the entry in this field has to be changed.

Other Configuration Fields

Diagram Files

Block History Files
DSP Chip Executable Files
System Source Templates
User Source Templates
Waveform Files

Block History Files

This field specifies the location of block History (.ini)files used by the software. If installation was not done with the default settings, the entry in this field has to be changed.

Other Configuration Fields

Diagram Files
DSP Chip Executable Files
System Source Templates
User Source Templates
User Source Templates
Waveform Files

DSP Chip Executable Files

This field specifies the location of DSP Chip Executable Files used by the software. If installation was not done with the default settings, the entry in this field has to be changed.

Other Configuration Fields

Block History Files

Block Symbol (Bitmap) Files
Diagram Files
System Source Templates
User Source Templates
Waveform Files

6.3.2 DSP/Analog Hardware Configuration

See DSP/Analog Hardware Configuration in Section 8.2.2, DSP/Analog Hardware Configuration.

6.3.3 DSP Engine Configuration

Options in this section specify the location and command line for the DSP Engine to be used with the software. The DSPower installation default settings are C:\HSMACRO\hsmacro.exe for Hypersignal and C:\MATLAB\matlab.exe for MATLAB

DSP Engine Configuration Fields

Hypersignal Command Line

This field specifies the command-line needed to invoke Hypersignal Macro / Acoustic if either one is being used as the DSP Engine. The

installation defaults to c:\hsmacro\hsmacro.exe for Hypersignal Macro and c:\hsacous\hsacous.exe for Hypersignal Acoustic.

Related Fields:

MATLAB Command Line

DSPower C Engine Command Line

Real-Time DSP Command Line

Run-Time Source Path

Hypersignal Run-Time Source Path

MATLAB Command Line

This field specifies the command-line needed to invoke MATLAB if it is being used as the DSP Engine. The installation defaults to c:\matlab\bin\matlab.exe

Related Fields:

Hypersignal Command Line

DSPower C Engine Command Line

Real-Time DSP Command Line

Run-Time Source Path

MATLAB Run-Time Source Path

DSPower C Engine Command Line

This field specifies the command-line needed to invoke the C Engine if it is being used as the DSP Engine. The installation defaults to c:\dspower

Related Fields:

Hypersignal Command Line

MATLAB Command Line

Real-Time DSP Command Line

Run-Time Source Path

C/C++Run-Time Source Path

Real-Time DSP Command Line

This field specifies the command-line needed to invoke any Real-Time DSP if it is being used as the DSP Engine. The installation defaults to c:\dspower

Related Fields:

Hypersignal Command Line

MATLAB Command Line

DSPower C Engine Command Line

Run-Time Source Path

Run-Time Source Path

This field specifies the location of generated source files for the corresponding DSP Engine. The

installation defaults to the root directory of the DSP Engine(s) being used.

Related Fields:

Hypersignal Command Line

MATLAB Command Line

DSPower C Engine Command Line

Real-Time DSP Command Line

6.3.4 Online Editor Configuration

This option allows configuration of the editor for use with DSPower. The default entry during installation is "C:\WINDOWS\notepad.exe"

Online Editor Command Line

This field specifies the command-line needed to invoke the online editor to be used with the software. The installation defaults to c:\windows\notepad.exe.

Other Options

Online Editor Starup Path

Online Editor Options

Online Editor Startup Path

Other Options

Online Editor Command Line

Online Editor Options

This section controls customization of the software to a certain extent. The available options for the editor are:

Save Position Between Use

Option to save editor window position between sessions.

Highlight DSPower Keywords

Allows highlighting of DSPower keywords in the editor.

Highlight Dataflow Connections

Allows highlighting of Dataflow Connections in the diagram.

Highlight Controlflow Connections

Allows highlighting Controlflow Connections in the diagram.

Other Options

Online Editor Command Line

Online Editor Startup Path

6.3.5 Online Debugger Configuration

Online Debugger Command Line

Other Options

Online Debugger Startup Path

Online Debugger Options

Online Debugger Startup Path

Other Options

Online Debugger Command Line

Online Debugger Options

Online Debugger Options

Save Position Between Use

Other Options

Online Debugger Command Line

Online Debugger Startup Path

6.3.6 Block Libraries Configuration

This dialog box lists all template / function files available in the system directory. It allows initialization of the Block Selector

dialog box. Controls include Listing / Unlisting, duplicating and Renaming of system template/function files.

Block List Criteria

This field specifies the list criteria. The following categories are available:

System Library

Lists specified file types from the system library. The system library is as specified in System Source Templates.

User Library

Lists specified files in the user library. The user library is as specified in User Source Templates.

Template files

Lists all Template Files in the selected directories.

Function Files

Lists all Function files in the selected directory.

Interface Files

.mac

Lists all .mac files in the selected directory.

.m

Lists all .m files in the selected directory.

.c / .cpp

Lists all .c / .cpp files in specified directory.

You can further chose the blocks you want to appear on the Block Selector using the select option. You can choose the Select All Button to choose all available options.

6.3.7 Operator Mode (Stand-Alone Apps) Configuration

Stand-Alone Window Text

Other Options

Operator Mode Configuration Options

Operator Mode Configuration Options

Dislplay DSP engine status/error message

Allow user positioning of control surface area

Other Options

Stand-Alone Window Text

6.4 Creating Block Diagrams

6.4.1 Diagrams

Diagrams are basic to DSPower. A diagram can be created from one or more blocks. Once these blocks have been run, the source code for that diagram will be generated and saved. If the same diagram needs to be run again, the file can be retrieved and re-executed and if this is used very often, making it part of a block library is suggested. It is also recommended that any customized blocks should be saved in the user library. The source code template with the defaults is stored in the system library and can be used to start over.

See Diagram Options for options to customize a diagram in DSPower

6.4.2 Connecting Blocks

Connecting blocks in DSPower is very easy. To make a connection, click on the output blue connector of the first block and click on the input connector of the second block. The line which appears shows the connection. The red inputs to each block can be used to determine the control flow.

6.4.3 Creating Projects

Projects can be created in DSPower with multiple diagrams. Saved projects have a .dpr extension.

6.4.4 Multiple Diagrams

Multiple diagrams ---> Introduction.

6.4.5 Saving and Retrieving Block Diagrams

Although initially not obvious as a user-interface feature, the way block-diagrams are saved in a "source code generation program" truly has bearing on the way you interact with such programs. One striking difference between the "source code generation user-interface" in DSPower and that in other programs: DSPower saves and retrieves block-diagrams directly in source code form.

6.4.6 Executing Block Diagrams

Also see Run One Block,

6.5 Adding User Defined Blocks

6.5.1 Creating and Adding Blocks

At any time, you can add a completely new block to a specific section of a block library, by supplying a source code template file and certain other information needed by DSPower.

6.5.2 Creating derivative Blocks

Similar to creating a procedure block, except that certain, specific system blocks are available that can be used as the original block.

6.5.3 Adding Code Blocks

Code blocks are blocks that contain arbitrary code that may not be easily categorized as function, displays, filter, etc. Typically, such blocks might create and manipulate global variables or arrays needed by the entire block diagram.

6.5.4 Modifying Controls

Inside source code template files, controls can be defined for both the parameter setup box and interactive display window, if any, for all blocks.

6.6 Interactive Operation

6.6.1 Run One Block

-How to specify menu

-filenames, drag-drop etc.

6.6.2 Loading Waveforms

See Waveform Locator, Dragging Waveform Files etc.

6.6.3 Loading Setup and Style Files

See Setup/Style Locator

6.6.4 Viewing and Editing Template and Generated Source Files

Also see Online Editor Configuration

6.7 Tools in DSPower

6.7.1 Operator Mode

6.7.2 Make Stand-Alone Icon

6.7.3 Block Wizard

6.7.4 Waveform Headitor

6.8 Debugging Diagrams

-How to find suspected problems

-disp blocks

-partial execution

-storing waveform files etc.

7 Using Engine Manager

Overview

Engine Manager is composed of the file enmgr.dll and support programs and files. It is an integral part of DSPower-HWLib and DSPower-SPLib software. Its main purpose is to provide control over DSP Engines, so that they can be opened and perform useful work more or less transparently to the calling application. Engine Manager can keep track of multiple engines, including multiple copies of the same engine, at one time. It generates callback messages when necessary, redirecting information to previously designated user window procedures. Engine Manager contains internal GetMessage loops, which are used when a synchronous call has been made and it must wait before returning results to a calling application.

For a concise description and explanation of individual calls

inside Engine Manager see Signalogic DSP Software Reference Guide section 2.2.1, Engine Manager Functions.

For more information about, and the basic philosophy underlying the engine concept, see DSP Engines, in Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components.

Main Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager

7.1 Engine Manager Access

[@k implicit loading, explicit loading]

To access Engine Manager, the enmgr.dll library must be visible to your application's calling program. The easiest way to do this is to include the enmgr.lib file in your project. For C/C++ projects, please see the example Borland C/C++ v3.1 and v4.5, and the Microsoft Visual C/C++ v1.5 and v2.0, project files, included with the DSPower-HWLib or DSPower-SPLib software. For MATLAB projects, please see the example .m files.

Once included in your project file, and thus linked into your final executable, the enmgr.lib file causes the Engine Manager DLL to be loaded and initialized when your executable first runs (actually, *before* any code in the executable runs). This is known as implicit loading.

Another way of accessing the enmgr.dll library is to load it from your application program dynamically, using the LoadLibrary call in the Windows API, followed by one or more GetProcAddress calls with specific function names as input parameters. This method is known as explicit loading, and is more complicated, requiring painstaking typecasting to create function prototype *pointers* which are set as a result of GetProcAddress calls. "Behind the scenes", this is also the technique used by MATLAB to load DLLs needed by calling .m programs--except that MATLAB simplifies the process by allowing DLL calls to look much like other MATLAB function calls.

Also, for C/C++ programs, the enmgr.h include file must be specified along with other include files that may be needed by the calling application. Note that the enmgr.h include file expects the standard Windows 3.1 windows.h include file to have been previously specified.

NOTE: For C/C++ development, make sure the include file directory

list in your IDE contains the DSPower-HWLib subdirectory. The example project files supplied with DSPower-HWLib software assume default include file subdirectories of "hwlib", "toollib", and "helplib" under the root-level installation subdirectory, although toollib and helplib are present only for demo purposes.

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager

7.2 Controlling DSP Engines

The main purpose of Engine Manager is to abstract the control of diverse DOS and Windows applications needed to perform DSP/analog hardware- or signal processing-related work. The core of the Engine Manager API consists of the following calls:

- DSEngineOpen
- DSEngineClose
- DSGetEngineUsage
- DSExecEngineFunction

- DSSendEngineCommand
- DSSetEngineWaveformPath
- DSSetEngineTemplatePath
- DSGetEngineErrorStatus
- DSRegisterEngineMsgWnd

For detailed information on these and other calls inside the Engine Manager API, including input parameters and output results, see Signalogic DSP Software Reference Guide section 2.2.1, Engine Manager Functions.

Related Topics

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager

7.2.1 Opening DSP Engines

Before many of the functions in DSPower-HWLib and DSPower-SPLib DLLs can be used, a DSP Engine must first be opened. In the source code example programs included with DSPower-HWLib and DSPower-SPLib software, a DSEngineOpen call is usually one

of the first few statements. Often, this call is made and the program has little further need to directly use the engine handle returned. Typical uses of the engine handle are:

- making DSAssignBoard calls in DSPower-HWLib
- executing DSP Engine functions using the DSExecEngineFunction call (see section 7.3 below, Executing Arbitrary Functions in DSP Engines)
- registering callback windows for various purposes
- closing engines using DSEngineClose (which is also done automatically by enmgr.dll when user-defined programs terminate, if needed)

The DSEngineOpen call opens DSP Engines such as Hypersignal-Macro series programs or MATLAB programs and places them in command mode, ready to accept instructions. DSP Engines are opened so as not to be visible (and conserve memory by not loading graphics drivers or other user-interface objects, if possible). The form of the call is as follows:

```
hEngine = DSEngineOpen(szEnginePath, szWaveformPath);
```

where szEnginePath and szWaveformPath are null-terminated strings (both input parameters are defined as LPCSTR). The return value, hEngine, is non-NULL on success, and NULL on failure. If the call fails, the DSGetEngMgrErrorStatus call can be used to get more information (see also section 7.5 below, Error Handling in Engine Manager). szEnginePath can be set to one of the following pre-defined

constants to conveniently specify supported DSP Engines:

szEnginePath	DSP Engine	Default Search Path

If one of the above constants is used, then the default subdirectories shown are used to attempt to locate the engine. For example, MATLAB is searched for as "x:\matlab\bin\matlab.exe", where "x" is the currently active drive. If instead the szEnginePath parameter contains a pointer to a valid string, then this string is assumed to contain explicit path information for the DSP Engine.

If the engine path is left NULL, then the DSEngineOpen function attempts to locate an appropriate DSP Engine, first trying Hypersignal-Macro software, then Hypersignal-Acoustic software, etc. This approach is used in the DSPower-HWLib demo programs, so these programs will run with whatever engine is installed on the system.

The szWaveformPath parameter can be set when a DSP Engine is open, or at any time using the DSSetEngineWaveformPath call. This path specifies where the DSP Engine will look when processing waveform files.

Internally, DSP Engines are opened and initialized using either a Windows WinExec call or functions inside shell.dll. If the WinExec call is used, the "show state" parameter is set to SW_HIDE, and in the case of DOS engines, a .pif file may be associated with the engine.

For more information, see also DSEngineOpen in Signalogic DSP Software Reference Guide section 2.2.1, Engine Manager Functions.

Related Topics

Sending DSP Engine Commands

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager

7.2.2 Sending DSP Engine Commands

From time to time, user-defined programs may need to send single commands or a stream of commands to an open DSP Engine. Being able to do so offers ultimate flexibility and may be the "last line of defense" in handling unusual situations. The DSSendEngineCommand call in enmgr.dll is intended for this

purpose. For example, to abort a MATLAB function taking a long time to process, you can stop the function using:

```
DSSendEngineCommand(hEngine, 3);
```

This sends the ctrl-C command to MATLAB. As another example, you can stop and restart an executing Hypersignal-Macro series software function without the overhead of submitting a new macro language source file by using:

```
DSSendEngineCommand(hEngine, 0x1b);
DSSendEngineCommand(hEngine, 0x0a);
```

The 0x1b value is equivalent to pressing the Escape key, and the 0x0a value is equivalent to the Ctrl-Enter key sequence. The above example might apply in a situation where you had changed a waveform file or filter coefficient file needed by the Hypersignal function, and then wanted to stop and restart the function causing it to use the new information.

Related Topics

Opening DSP Engines

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager

7.3 Executing Functions in DSP Engines

Engine Manager has a general-purpose DSExecEngineFunction function that allows DSP Engine functions to be invoked from user-defined programs. This facility is not to be confused with DSP Engine functions, or combinations of functions, which are executed automatically and transparently as a result of standard API calls inside DSPower-HWLib or DSPower-SPLib DLLs.

The purpose of the DSExecEngineFunction call is to allow engine functions to be called directly, assuming knowledge of the syntax and parameters of the engine call. Input parameters are the engine handle and a string, which should define the complete engine command. For example, to evaluate a difference equation on a waveform file with Hypersignal-Macro as the engine, a statement like the following might be used:

```
DSExecEngineFunction(hEngine, "f_diffeq(\"sp_avg\", \"y[n] = y[n] + (x1[n] – x1[n-M])/Fr\", \"speech\", \"\", 0, 0, 0)", uMode);
```

Note the use of \" in the above calls when quote marks are needed inside a C/C++ language string already differentiated by quote marks. Note also the hEngine parameter, which should represent a valid engine handle from a previous DSEngineOpen call, and the uMode parameter, which specifies either synchronous or

asynchronous operation. Synchronous operation means the calling application waits for the engine function to complete; asynchronous operation means the calling application continues to execute simultaneously. uMode values of DS_EEF_SYNC and DS_EEF_ASYNC are predefined inside the enmgr.h include file.

Currently, engine functions cannot be queued via repeated calls to DSExecEngineFunction. However, this same functionality can be achieved by queuing calls in a local structure defined and maintained in the calling application, and then by emptying the queue in response to program done messages. See Program Status Messages from DSP Engines, in section 7.4 below, Registering Callback Messages from DSP Engines.

As another example, to call a simple FFT inside MATLAB, a statement like the following might be used:

```
DSExecEngineFunction(hEngine, "x = f_load(\"speech.wav\")", DS_EEF_SYNC);
DSExecEngineFunction(hEngine, "y = fft(x)", DS_EEF_ASYNC);
```

Note that some method must be utilized to initialize arrays before being used as input to MATLAB functions. The above example shows the DSPower series software f_load function being used to load the time domain waveform file "speech.wav". Note also that the first function has been executed synchronously, which means the function does not return until complete. The second function has

been executed asynchronously, which means that the calling application continues to execute after the function has been started.

NOTE: If a callback window has been registered using the DSRegisterEngineMsgWnd function, a "program done" message will be sent to this window when the engine function specified in a DSExecEngineFunction call completes. See Program Status Messages from DSP Engines, in section 7.4 below, Registering Callback Messages from DSP Engines.

For more information on DSPower series software functions, see Signalogic DSP Software Reference Guide section 2.1, DSP Engine Functions. For more information on MATLAB functions, see the MATLAB Reference Guide and the Signal Processing Users Guide (The MathWorks).

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager

7.4 Registering Callback Messages from DSP Engines

Engine Manager allows user-specified window or control handles to be "registered" to several types of DSP Engine callback messages, using the DSRegisterEngineMsgWnd function. Once registered, window or control handles receive callback messages at various times, depending on various circumstances.

Below are the current callback message types available in DSPower-HWLib and DSPower-SPLib software:

- engine status messages
- program status messages
- buffer ready messages.
- semaphore messages
- statistics and parameter update messages

A window or control handle must be registered for each of the above callback message types before they can be sent by DSP Engines. Engine and program status messages are handled by Engine Manager (enmgr.dll), and buffer ready, semaphore, and statistics and parameter update messages are handled by either DSPower-HWLib (hwlib.dll) or DSPower-SPLib (splib.dll).

Below are some examples of using the DSRegisterEngineMsgWnd function:

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETSTATISTICSMSG |
DS_REMW_SETLEVELMSG, hwndCtl);
```

In the above call, the DS_REMW_SETSTATISTICSMSG and DS_REMW_SETLEVELMSG constants have been used to indicate that amplitude level and statistics and parameter update messages should be sent to hwndCtl. For instance, hwndCtl could be a static control in a dialog box used to display text information, or a graphical meter control. Both of these examples plus more are shown in the source code file for the digital tape recorder demo included with DSPower-HWLib software (dtape.cpp).

Note in the above example that callback message type constants can be combined. There is also a DS_REMW_SETALLMSG constant which can be used to register one window for all callback message types. Below is another example, which would apply in the case of executing general engine functions with the DSExecEngineFunction call:

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETPROGSTATUSMSG,
hwnd);
```

In the above call, the DS_REMW_SETPROGSTATUSMSG constant has been used to indicate that program status messages should be sent to hwnd.

For example, if the DSExecEngineFunction call was to be used to start an engine function or program, then the calling application might first specify that hwnd should receive a WM_DSPENGINE_PROGSTATUS message indicating completion of the function or program.

Below is an example which could be used in the case of processing continuous analog input data from DSP/analog hardware:

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG,
hwnd);
```

In the above call, the DS_REMW_SETDSPDATARDYMSG constant has been used to indicate that buffer ready and semaphore ready messages available from DSP Engines via DSPower-HWLib should be sent to hwnd. See Example C/C++ Code: Processing Buffer Ready Messages in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software, for a complete example showing a typical response to buffer ready messages.

Below are the possible callback message types for which window or control handles can be registered:

Update Message Category	Summary of Meaning
DS_REMW_	

SETENGINESTATUSMSG

DS_REMW_SETPROGSTATUSMSG

DS_REMW_SETDSPDATARDYMSG

DS_REMW_SETSTATISTICSMSG

DS_REMW_SETBUFNUMMSG

DS_REMW_SETLEVELMSG

DS_REMW_SETALLMSG

For the most complete source code examples of registering and processing the above callback message types, please see the dtape.cpp C/C++ file included with DSPower-HWLib software.

Callback Message Types

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

DSPower-HWLib Function Overview

Creating Programs for DSPower-HWLib and DSPower-SPLib

Acquiring Data Using DSPower-HWLib Software

Status Messages from DSP Engines

Engine status messages are sent by DSP Engines via Engine Manager regarding general operations and error conditions. No messages are sent unless user-specified window or control handles are registered using the DSRegisterEngineMsgWnd function. Below are the current engine status messages possible:

Engine Status Message	Summary of Meaning	wParam	lParam

The lParam parameter in the error messages listed above points to an ERRORMSGINFO structure, which contains information about the error, including error number and description, error log filename, etc. The calling application does not need to return a specific value

after processing engine status messages.

For detailed information about the above messages, see Signalogic DSP Software Reference Guide section 6.1, DLL Messages.

See the enmgr.h file for the current actual value of these messages. Normally, program status messages are defined as WM_USER + n, where n is a constant and WM_USER is defined in windows.h. Other WM_DSPENGINE messages are defined similarly.

Callback Message Types

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Program Status Messages from DSP Engines

Program status messages are sent by DSP Engines as a result of the calling application invoking a high-level function, such as DSExecEngineFunction or DSAcquireWvfrmFile. Messages are only sent if the calling application has previously registered a window or control handle using the DSRegisterEngineMsgWnd function.

Current program status messages in DSPower-HWLib and DSPower-SPLib software are listed below:

Program Status Message	Summary of Meaning	wParam	lParam

For detailed information about the above messages, see Signalogic DSP Software Reference Guide section 6.1, DLL Messages.

See the enmgr.h file for the current actual value of these messages. Normally, program status messages are defined as WM_USER + n, where n is a constant and WM_USER is defined in windows.h. Other WM_DSPENGINE messages are defined similarly.

Callback Message Types

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

DSPower-HWLib Function Overview

Creating Programs for DSPower-HWLib and DSPower-SPLib

Acquiring Data Using DSPower-HWLib Software

Buffer Ready Messages from DSP Engines

Buffer ready messages are sent by DSP Engines when a pre-specified buffer number is set, indicating that another analog input buffer has been acquired or another analog output data buffer is needed from the calling application. Reserved values can be used to indicate that buffer ready messages should be generated by a DSP Engine as a result of native acquisition, digital oscilloscope, or other operating modes which have been initialized and started by the calling application. They can also be generated by user-defined DSP code using either polled or interrupt-driven notification. An example of the former might be buffer switching for continuous

analog input or output transfer between disk file.

Buffer ready messages are set up when an application calls DSWaitForBuffer. Below is a source code example:

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForBuffer(hBoard, nBufferNum, dwFlagAddr, uNotifyType);
```

In the above calls, hwnd is first registered to receive buffer ready and semaphore ready messages from the DSP Engine; in this case, hwnd will receive WM_DSPENGINE_BUFRDY messages (see below). hBoard is a handle to a DSP/analog hardware type created by DSAssignBoard, nBufferNum is the buffer number to wait for, dwFlagAddr is the address in DSP memory containing the buffer number, and uNotifyType is the type of notification for the engine to use.

nBufferNum values 0 through 15 are reserved for standard DSP Engine high-level function "data nodes", which are pre-defined sources and sinks of analog input and output data. In these cases, dwFlagAddr and uNotifyType are ignored. For example, nBufferNum values of 0 and 1 indicate continuous analog input data, alternating between buffers 0 and 1. uNotifyType is ignored for reserved nBufferNum values because DSP Engines may use specialized methods of buffer-checking, which may involve hardware register bits or other optimized techniques (not

necessarily DSP board memory). The concept behind reserved data nodes is that more than one calling application may register a window handle to receive the same buffer ready messages.

Further defined are subpairs of reserved nBufferNum values corresponding to nodes which use a dual-buffer switching mechanism for processing continuous data. nBufferNum values of 0 and 1 indicate continuous analog input data, alternating between buffers 0 and 1, nBufferNum values of 2 and 3 indicate continuous analog output data, values of 4 and 5 indicate telephone adapter input data, etc. For example, if operating Hypersignal-Macro series software in its digital oscilloscope mode, then the first DSWaitForBuffer call would specify buffer 0. After receiving this buffer, another DSWaitForBuffer call would be issued with buffer = 1, then again with buffer = 0, etc.

Values for nBufferNum not equal to 0 through 15 can be used to set up user-defined buffer ready messages. In these cases, dwFlagAddr and uNotifyType must be specified. Note that if the initial value at dwFlagAddr should not be zero, then a DSPutMem call should be performed before DSWaitForBuffer to initialize the flag value correctly.

NOTE: In the case of user-defined buffer ready messages, buffer numbers and memory flag locations

used must be "agreed to" beforehand by the calling application and the executing DSP code.

Currently, values for uNotifyType can be DS_WFB_POLLED and DS_WFB_INT, which specify polling and interrupt-driven notification, respectively. If a uNotifyType value of DS_WFB_POLLED is specified, then the DSP Engine will periodically check to see if the specified buffer is ready, and cause WM_DSPENGINE_BUFRDY messages to be sent to hwnd as needed. Currently, if uNotifyType is DS_WFB_POLLED, then only one DSWaitForBuffer call can be active at one time for each active DSP Engine. To establish multiple DSWaitForBuffer requests at one time in polled mode, more than one DSP Engine handle should be active; i.e. more than one copy of .the DSP Engine open simultaneously.

If a uNotifyType value of DS_WFB_INT is specified, the DSP Engine will set up and enable the next available DSP-to-host PC interrupt. When the interrupt occurs, a WM_DSPENGINE_BUFRDY message is sent to hwnd. Note that if uNotifyType is DS_WFB_INT, then more than one DSWaitForBuffer call can be established at one time for each active DSP Engine. Note also that for user-defined buffers using

DS_WFB_INT, the dwFlagAddr location is checked to make sure it has changed from its initial value.

Current buffer ready messages in DSPower-HWLib and DSPower-SPLib software are listed below:

Buffer Ready Message	Summary of Meaning	wParam	lParam

The lParam parameter points to a DSPDATARDYMSGINFO structure, which contains

information about the buffer, including buffer number, board handle, and flag address and value (if any). The calling application does not need to return a specific value after processing buffer ready messages.

For detailed information about the above messages, see Signalogic DSP Software Reference Guide section 6.1, DLL Messages.

See the enmgr.h file for the current actual value of these messages. Normally, WM_DSPENGINE_BUFRDY is defined as WM_USER (defined in windows.h), and subsequent messages are incremented in value.

Callback Message Types

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines
Registering Callback Messages from DSP Engines
Error Handling in Engine Manager
DSPower-HWLib Function Overview
Creating Programs for DSPower-HWLib and DSPower-SPLib
Acquiring Data Using DSPower-HWLib Software

Semaphore Messages from DSP Engines

Semaphore messages are sent by DSP Engines when a pre-specified flag value is set, indicating that a DSP memory semaphore (flag) has changed. Reserved values can be used to indicate that semaphore ready messages should be generated by a DSP Engine as a result of native acquisition, digital oscilloscope, or other operating modes which have been initialized and started by the calling application. They can also be generated by user-defined DSP code using either polled or interrupt-driven notification. An example of the former might be a "done", or "not busy" status for DSP accelerated processing.

Semaphore messages are set up when an application calls DSWaitForFlag. Below is a source code example:

```
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForFlag(hBoard, nFlagNum, dwFlagAddr, uNotifyType);
```

In the above calls, hwnd is first registered to receive buffer ready and semaphore ready messages from the DSP Engine; in this case,

hwnd will receive WM_DSPENGINE_FLGRDY messages (see below). hBoard is a handle to a DSP/analog hardware type created by DSAssignBoard, nFlagNum is the number of the flag to wait for, dwFlagAddr the address of the flag's location in DSP board memory, and uNotifyType is the type of notification for the engine to use.

For example, if a flag was initialized to zero, then a DSWaitForFlag call could be used to cause the DSP Engine to send a WM_DSPENGINE_FLGRDY message when the value of the flag became non-zero.

nFlagNum values 0 through 15 are reserved for standard DSP Engine high-level function "semaphore nodes", which are pre-defined status indicators for standard types of analog input and output data and acceleration processing. In these cases, dwFlagAddr and uNotifyType are ignored. For example, an nFlagNum value of 0 indicates accelerated file-based time domain data processing. uNotifyType is ignored for reserved nFlagNum values because DSP Engines may use specialized methods of flag-checking, which may involve hardware register bits or other optimized technique (not necessarily DSP board memory). The concept behind reserved semaphore nodes is that more than one calling application may register a window handle to receive the same semaphore messages.

Values for nFlagNum not equal to 0 through 15 can be used to set up user-defined semaphore messages. In these cases, dwFlagAddr and uNotifyType must be specified. Note that if the initial value of the semaphore should not be zero, then a DSPutMem call should be performed before DSWaitForFlag to initialize the flag value correctly.

NOTE: In the case of user-defined semaphore messages, flag values and memory locations used must be "agreed to" beforehand by the calling application and the executing DSP code.

Currently, values for uNotifyType can be DS_WFB_POLLED and DS_WFB_INT, which specify polling and interrupt-driven notification, respectively. If a uNotifyType value of DS_WFB_POLLED is specified, then the DSP Engine will periodically check to see if the specified semaphore is ready, and cause WM_DSPENGINE_FLGRDY messages to be sent to hwnd as needed. Currently, if uNotifyType is DS_WFB_POLLED, then only one DSWaitForFlag call can be active at one time for each active DSP Engine. To establish multiple DSWaitForFlag requests at one time in polled mode, more than one DSP Engine handle should be active; i.e. more than one copy of .the DSP Engine open simultaneously.

If a uNotifyType value of DS_WFB_INT is specified, the DSP Engine will set up and enable the next available DSP-to-host PC interrupt. When the interrupt occurs, a WM_DSPENGINE_FLGRDY message is sent to hwnd. Note that if uNotifyType is DS_WFB_INT, then more than one DSWaitForFlag call can be established at one time for each active DSP Engine. Note also that for user-defined semaphores using DS_WFB_INT, the dwFlagAddr location is checked to make sure it has changed from its initial value.

The current semaphore messages in DSPower-HWLib and DSPower-SPLib software are listed below:

Semaphore Message	Summary of Meaning	wParam	lParam

The lParam parameter points to a DSPDATARDYMSGINFO structure, which contains information about the semaphore, including flag number, board handle, and flag address and value. The calling application does not need to return a specific value after processing semaphore messages.

For detailed information about the above messages, see Signalogic DSP Software Reference Guide section 6.1, DLL Messages.

See the enmgr.h file for the current actual value of these messages. Normally, WM_DSPENGINE_FLGRDY is defined as WM_USER (defined in windows.h), and subsequent messages are incremented in value.

Callback Message Types

Status Messages from DSP Engines

Program Status Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

DSPower-HWLib Function Overview

Creating Programs for DSPower-HWLib and DSPower-SPLib

Acquiring Data Using DSPower-HWLib Software

Update Messages from DSP Engines

In DSPower-HWLib, update messages are associated with high-level functions of which the engine is capable. For these functions, the calling application has instructed the DSP Engine to perform and complete--unaided--a complicated task, such as acquiring a waveform file, with only initial instructions being given. Update messages are typically sent when a parameter or

other data value has been generated or updated by the DSP Engine as a result of the high-level function which has been invoked by the calling application. Examples include the maximum amplitude in a buffer of acquisition data, a dBA value from a spectrum analyzer function, or current a sample counter in a data logging operation. Some update messages may be sent one-time or sporadically; others may be sent on a regular basis.

A specific example might be the sampling frequency used by the DSP/analog hardware during a DSAcquireWvfrmFile function call. Although a sampling frequency was probably specified in the initial call instructions, the DSP/analog hardware may only allow certain physical values. Once the engine has determined which allowed value is nearest to the desired value, it sends an update message indicating the actual value that will be used. It is up to the calling application to do something with this data or discard it.

A window or control handle must be "registered" to one or more update message types using DSRegisterEngineMsgWnd, before messages can be received. Below is a source code example:

There are several possible update messages which can be sent by DSP Engines. Included are error and status messages, messages related to high-level functions, and user-defined messages originating

from DSP code. Some of these messages are listed below:

Update Message	Summary of Meaning	wParam	lParam

For detailed information about the above messages, see Signalogic DSP Software Reference Guide section 6.1, DLL Messages.

See the enmgr.h file for the current actual value of these messages. Normally, WM_DSPENGINE_BUFRDY is defined as WM_USER (defined in windows.h), and subsequent messages are incremented in value.

Callback Message Types

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

Update Messages from DSP Engines

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

DSPower-HWLib Function Overview

Creating Programs for DSPower-HWLib and DSPower-SPLib

Acquiring Data Using DSPower-HWLib Software

7.5 Error Handling in Engine Manager

The Engine Manager error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into enmgr.dll fail, there are two levels of error possibilities: the DLL itself (or "base level"), and engine-related. The DSGetEngMgrErrorStatus and DSGetEngineErrorStatus calls can be used to attempt to learn more information about the most recent failure.

Engine Manager also maintains a "status listbox" which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of *both status and error* text displays resulting from all Engine Manager calls.

Related Topics

7.5.1 DLL Error Handling in Engine Manager

Engine Manager has a "base level" error/status routine which can be used to look for general DLL error conditions not necessarily associated with a specific engine. Below is an example showing further investigation after a DSEngineOpen failure:

```
hEngine = DSEngineOpen(szEnginePath, szWaveformPath);
if (!hEngine) {
    nErrorType = DSGetEngMgrErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSEngineOpen would be NULL in the case of failure, since a non-NULL value would be a valid engine handle. Thus, to find out more information, the DSGetEngMgrErrorStatus call is used.

Below is a partial list of error values returned by DSGetEngMgrErrorStatus, along with a brief description of their

meaning. These constants can be found partly in the enmgr.h file and partly in the dllerr.h file; for a complete list all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions.

dllerr.h or enmgr.h constant	Summary of Meaning
DSCOULDNOTOPENENGINE	

DSCOULDNOTCREATEBKGNDWINDOW

DSENGINETIMEOUT

DSCOULDNOTSTARTBKGNDTASK

DSNOBKGNDWINDOW

D

SCOULDNOTINITIALIZEDLL

Note that in C/C++ programs, it is safest to treat the above error

definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions, for more information about this grouping.

Related Topics

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

7.5.2 Engine Error Handling

When calls into enmgr.dll, hwlib.dll, or splib.dll which directly or indirectly involve an assigned DSP Engine handle fail, DSGetEngineErrorStatus can be used to attempt to learn more information about the failure. This approach allows calls to be treated with logical (boolean) tests inside C/C++ if statements; for example:

```
if (!hBoard = DSAssignBoard(hEngine, szBoardDesignator, uBusType,
wIOBaseAddr, wMemBaseAddr)) {
```

```
    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
...continue with valid hBoard...
```

In the above example, if DSAssignBoard is successful, then the hBoard parameter will contain a non-zero value representing the newly assigned board handle. Note that the "base-level" DSGetHWLibErrorStatus routine could also be checked, in case the problem *was not* engine related.

Below is another example, using the DSInitBoard call found in DSPower-HWLib (hwlib.dll):

```
if (!DSInitBoard(hBoard)) {

    if (DSGetEngineErrorStatus(hEngine) ==
    DSHARDWARENOTRESPONDING) {

        MessageBox(NULL, "DSP/analog hardware not responding; are I/O base
        address jumpers set correctly?", "Cool DSP Application #1", MB_OK |
        MB_ICONEXCLAMATION);
    }
}
```

Note that in the above example, an engine handle (hEngine) was implied, since a successful DSAssignBoard call must have been made before attempting a DSInitBoard call.

Below is a partial list of error values returned by DSGetEngineErrorStatus, along with a brief description of their meaning. These constants can be found in the enmgr.h file; for a complete list all DSPower series software error conditions by error number, see Error and Status

Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions.

enmgr.h constant	Summary of Meaning
	Typically means the DSP Engine executable program file could not be located.
	An invalid en

gine path was specified in a DSEngine OpenCall.

Time-out limits were exceeded in waiting for the DSPEn

gi
ne
to
su
cc
ess
ful
ly
op
en
an
d
ini
tia
liz
e,
or
res
po
nd
to
a
co
m
ma
nd
or
re
qu
est
.

Th
e
Pi
pe.
38
6
Vx
D
co
ul
d
no

t
be
op
en
ed
for
a
D
O
S
D
SP
En
gi
ne.
M
ak
e
sur
e
tha
t
pi
pe.
38
6
is
list
ed
in
the
"3
86
En
h"
se
cti
on
of
the
sy
ste
m.

ini
fil
e
on
the
x:\
wi
nd
ow
s
su
bd
ire
cto
ry.

Th
e
D
SP
En
gi
ne
fai
led
to
ini
tia
liz
e
co
rre
ctl
y.

A
cal
l
wa
s
ma
de
re
qu

iri
ng
an
en
gi
ne
ha
nd
le
bu
t
no
en
gi
ne
s
we
re
op
en.

A
cal
l
wa
s
ma
de
usi
ng
an
en
gi
ne
ha
nd
le
wh
ich
di
dn'
t
sp
eci

fy an engine currently being maintained by enmgr.dll.

A call was made using a NULL engine handle.

A cal

l
wa
s
ma
de
usi
ng
an
en
gi
ne
ha
nd
le
wh
ich
po
int
ed
to
me
m
or
y
alr
ea
dy
dis
car
de
d.

Note that in C/C++ programs, it is safest to treat the above error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions, for more information about this grouping.

Related Topics

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

7.5.3 Error/Status Window in Engine Manager

The Engine Manager DLL (enmgr.dll) maintains an error and status listbox window at all times. This window can be shown or hidden at any time using the DSShowEngMgrStatusWindow or DSHideEngMgrStatusWindow calls. The Hardware Manager DLL (hwmgr.dll) and DSPower-HWLib and DSPower-SPLib DLLs (hwlib.dll and splib.dll) also make use of this status listbox, by invoking the DSAddEngMgrStatusLine call. This call is exported from the Engine Manager DLL; its prototype can be made available by including enmgr.h. Calling applications including enmgr.h and adding enmgr.lib to their project file can enable the error and status listbox

as a debug tool or for operator feedback purposes, and add status listbox lines as needed. For more information, see sections 4.2.2 and 4.2.3, Include Files for DSPower-HWLib and DSPower-SPLib, and .LIB Files for DSPower-HWLib and DSPower-SPLib, respectively.

If displayed, the Engine Manager error and status listbox will show error and status reports not only from DSPower series DLLs, but also DSP Engines. Error and status reports are mixed together, being added to the listbox in chronological order, as they are received.

The position and size of the status and error listbox window is remembered by enmgr.dll in the "enmgr.ini" file, which is stored on the subdirectory from where enmgr.dll was loaded (normally, this is x:\windows\system, where "x" is the DSPower-HWLib or DSPower-SPLib installation drive.).

When the error and status listbox window is displayed, it is visible to whoever is using the application calling enmgr.dll functions. While this may be appropriate during test and development, it may not be for normal operation. One suggestion might be to enable a non-obvious keyboard switch which causes the listbox to become visible, to allow for greater detail in operator-reported errors.

If visible, the Engine Manager error and status listbox can be sized and

scrolled like any listbox. All error and status reports added to the listbox are saved, and older reports can be scrolled into view as needed. Engine Manager makes no attempt to monitor the number of status lines added to the list box, so eventually the oldest previous lines will "scroll off".

See Also

Error/Status Window in Engine Manager: Adding a Line

Error/Status Window in Engine Manager: Showing/Hiding

Related Topics

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager: Adding a Line

To add a line to the error and status listbox, use the following:

```
hwndStatus = DSAddEngMgrStatusLine(szText);
```

where szText is a null-terminated string (LPCSTR) and hwndStatus is the handle of the status window. Remember the line is added whether the listbox is visible or not.

See Also

Error/Status Window in Engine Manager: Adding a Line

Error/Status Window in Engine Manager: Showing/Hiding

Related Topics

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

Error/Status Window in Engine Manager: Showing/Hiding

The default state of the Engine Manager error/status listbox is hidden. To make the listbox visible, use:

```
DSShowEngMgrStatusWindow();
```

and to make it not visible, use:

```
DSHideEngMgrStatusWindow();
```

Note that initially the error and status listbox is displayed on top of other windows, but it does not have

the "always-on-top" (WS_EX_TOPMOST) attribute, so it can easily be covered by operator activity. If you need the listbox to always stay on top, make the following Windows calls:

```
SetWindowLong(hwndStatus, GWL_EXSTYLE, GetWindowLong(hwndStatus, GWL_EXSTYLE) | WS_EX_TOPMOST);
UpdateWindow(hwndStatus);
```

where hwndStatus is the handle of the error and status listbox returned from a DSAddEngMgrStatusLine call.

See Also

Error/Status Window in Engine Manager: Adding a Line

Error/Status Window in Engine Manager: Showing/Hiding

Related Topics

DLL Error Handling in Engine Manager

Engine Error Handling

Error/Status Window in Engine Manager

General Topics

Engine Manager Access

Controlling DSP Engines

Executing Functions in DSP Engines

Registering Callback Messages from DSP Engines

Error Handling in Engine Manager

8 Using Hardware Manager

Overview

This section provides information about the Hardware Manager, a DLL (hwmgr.dll) which provides the following capabilities:

- maintains an ASCII text Hardware Setup file, which contains detailed, specific information about one or more DSP/analog hardware types
- maintains mapping of logical "hardware variables" to currently specified physical "hardware values" for use by other Signalogic DSP Software components
- provides access to the Hardware Setup File, and to current logical hardware variables and physical hardware values through an API
- generates and displays on demand the Hardware Selector dialog box, which allows DSP/analog hardware to be selected from a list box and configured by the user

The Hardware Manager DLL is included with DSPower-HWLib and DSPower-Block Diagram software. However, most of the flexibility and customization features of the Hardware Manager arise from its use with DSPower-

HWLib software. For example, the Hardware Setup file can easily be customized to display only certain hardware types, or to application-specific hardware descriptions, using a standard text editor. See section 8.1.2 below, Hardware Setup File.

For information about DSPower-HWLib software, see chapter 4, Using DSPower-HWLib and DSPower-SPLib Software.

Main Topics

Hardware Manager Access

Hardware Selector

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

Hardware Setup File

Error Handling in Hardware Manager

8.1 Hardware Manager Access

[@k implicit loading, explicit loading]

To access Hardware Manager, the hwmgr.dll library must be visible to your application's calling program. The easiest way to do this is to include the hwmgr.lib file in your project. For C/C++ projects, please see the example Borland C/C++ v3.1 and v4.5, and the Microsoft Visual C/C++ v1.5 and v2.0, project files, included with DSPower-

HWLib software. For MATLAB projects, please see the example .m files. Once included in your project file, and thus linked into your final executable, the enmgr.lib file causes the Engine Manager DLL to be loaded and initialized when your executable first runs (actually, *before* any code in the executable runs). This is known as implicit loading.

Another way of accessing the hwmgr.dll library is to load it from your application program dynamically, using the LoadLibrary call in the Windows API, followed by one or more GetProcAddress calls with specific function names as input parameters. This method is known as explicit loading, and is more complicated, requiring painstaking typecasting to create function prototype *pointers* which are set as a result of GetProcAddress calls. "Behind the scenes", this is also the technique used by MATLAB to load DLLs needed by calling .m programs--except that MATLAB simplifies the process by allowing DLL calls to look much like other MATLAB function calls.

Also, for C/C++ programs, the hwmgr.h include file must be specified along with other include files that may be needed by the calling application. Note that the hwmgr.h include file expects the standard Windows 3.1 windows.h include file to have been previously specified.

NOTE: For C/C++ development, make sure the include file directory list in your IDE contains the DSPower-HWLib subdirectory. The example project files supplied with DSPower-HWLib software assume default include file subdirectories of "hwlib", "toollib", and "helplib" under the root-level installation subdirectory, although toollib and helplib are present only for demo purposes.

General Topics

Hardware Selector

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

Hardware Setup File

Error Handling in Hardware Manager

8.2 Hardware Selector

In DSPower-Block Diagram software, the Hardware Selector is activated by selecting Configuration | DSP/Analog Hardware. The Hardware Selector dialog box should then appear.

When using DSPower-HWLib software, the Hardware Selector is activated when a calling program makes the DSShowHardwareSelectorDlg call into hwmgr.dll. The prototype for this call is made available by including the hwmgr.h file.

Upon activation, the Hardware Selector dialog box displays the following:

- A hardware type listbox, which lists possible hardware selections alphabetically by manufacturer.
- A hardware configuration section, which allows configuration of important hardware parameters, including I/O and/or memory base addresses, host PC bus type, default executable DSP file, processor speed, and more. The current contents of the hardware configuration section reflect the current hardware type listbox selection, and change when the selected item in the listbox changes.
- A "hardware variable" section, which allows mapping of hardware variables commonly used in DSPower-Block Diagram software functions to different hardware types.

Note that the selections available in the hardware type listbox, and thus what can appear in the hardware configuration section, is entirely determined by the current contents of the Hardware Setup file (hwsetup.lst), which is maintained by the Hardware Manager DLL (hwmgr.dll). See section 8.3 below, Hardware Setup File, for more information.

Related Topics

General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

Error Handling in Hardware Manager

8.2.1 Selecting DSP/Analog Hardware Types

After activating the Hardware Selector (see section 8.2 above, Using the Hardware Selector), a dialog box is displayed that includes a listbox showing possible hardware selections. The contents of the listbox select the current contents of the Hardware Setup file (hwsetup.lst). In the Hardware Setup file, hardware entries are organized in groups by manufacturer; in the listbox, the manufacturer name is prefixed to the hardware type, causing all hardware selections to be listed alphabetically by manufacturer.

When selecting a hardware type, note that there can be one or more closely related selections for the same basic hardware. For example, the Ariel PC-56D board can be selected to use its onboard analog I/O, or for use with Ariel's Digital Microphone, ProPort, or DatPort. Thus, keep the following in mind:

If you don't see your selection right away, make sure you check all possibilities for that manufacturer.

Check closely to make sure you have the right selection--even a small mistake can make the difference between a hardware

selection working correctly or not.

After highlighting (selecting) the desired hardware type, you should see its corresponding configuration parameters displayed in the hardware configuration section of the Hardware Selector dialog box. This is the area directly below the hardware type listbox. For a rundown of these options, see section 8.2.2 below, DSP/Analog Hardware Configuration.

Related Topics

DSP/Analog Hardware Configuration

Hardware Values and Variables

General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

Error Handling in Hardware Manager

8.2.2 DSP/Analog Hardware Configuration

When the Hardware Selector dialog box is displayed, it can be divided into three major areas:

- A hardware type listbox, which lists possible hardware selections alphabetically by manufacturer. See section 8.2.1 above, Selecting DSP/Analog Hardware Types.
- A hardware configuration section (underneath the hardware type listbox), which allows configuration of important hardware parameters, including I/O and/or memory base addresses, host PC bus type, default executable DSP file,

processor speed, and more. See the description of selectable items below.

- *For DSPower-Block Diagram software only*, an additional "hardware variable" dialog box, which can be invoked using the "Set Hardware Variables" button. This dialog box allows mapping of hardware variables commonly used in DSPower-Block Diagram software functions to different hardware values. See section 8.2.3 below, Hardware Variables and Values.

Whenever parameters are changed in the hardware type listbox or hardware configuration sections of the Hardware Selector and the OK button is clicked, changes are saved to the Hardware Configuration file (see section 8.3, Hardware Configuration File). Changes are discarded if the Cancel button is clicked.

See below for descriptions of the fields inside the hardware configuration section of the Hardware Selector.

Hardware Configuration Parameters

Board Designator

I/O Base Address

Memory Base Address

Processor Clock

Voltage Ranges

Number of Processors

DSP Program File

Analog Clock Type

Bus Type

Memory Architecture

Related Topics

Selecting DSP/Analog Hardware Types

Hardware Values and Variables

General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

Error Handling in Hardware Manager

Board Designator

[@j board designator string]

Board designators are short string descriptions specifying hardware types, typically comprised of a manufacturer abbreviation and DSP device abbreviation. At any time, the designator shown reflects the DSP/analog hardware type currently selected. Board designators cannot be edited directly in the Hardware Selector; modification, if necessary, can be achieved by modifying the Hardware Setup File using the DSSetHWMgrEntry function or directly editing it. Note, however, that changing board designators or adding new ones is not normally recommended. See Board Designator Field in Hardware Setup File, in section 8.3.1 below, Hardware Setup File Fields.

Hardware Configuration Parameters

Board Designator

I/O Base Address
Memory Base Address
Processor Clock
Voltage Ranges
Number of Processors
DSP Program File
Analog Clock Type
Bus Type
Memory Architecture

I/O Base Address

For DSP/analog hardware with host PC interfaces using an I/O-mapped addressing scheme, this field should be set to match the base I/O address established by the jumpers or switch(es) on the hardware. Typical examples are 300H, 290H, 340H, etc. Values should include the 'H' suffix if entered in hexadecimal; values can also be entered in "C" hexadecimal notation (e.g. 0x300, 0x290, 0x340).

Note that some hardware do not actually include jumpers or switches; but in these cases, selecting an appropriate I/O base address value still needs to be done correctly.

Nearly all hardware types use an I/O base address even if a memory base address. Typically, there may actually be from 8 to 32 individual I/O locations offset from the base address. If a Hypersignal-Macro

Series software package is being used as a DSP Engine, see chapter 3 in the Hypersignal-Macro Series Hardware Reference Guide for information about I/O base addresses for specific hardware types.

For more information about I/O address conflicts which can arise with other installed hardware in the host PC, see Signalogic DSP Software System Guide section 4.3, Troubleshooting DSP/Analog Hardware Problems.

Hardware Configuration Parameters

Board Designator

I/O Base Address

Memory Base Address

Processor Clock

Voltage Ranges

Number of Processors

DSP Program File

Analog Clock Type

Bus Type

Memory Architecture

Memory Base Address

For DSP/analog hardware with a host PC interface using an memory-mapped addressing scheme, this field should be set to match the base memory address established by the jumpers or switch(es) on the hardware. Typical examples are D000H,

DC00H, DE00H, etc. Values should include the 'H' suffix if entered in hexadecimal; values can also be entered in "C" hexadecimal notation (e.g. 0xd000, 0xdc00, 0xde00).

Note that some hardware do not actually include jumpers or switches; but in these cases, selecting an appropriate memory base address value still needs to be done correctly.

Some, but not all, hardware types use a memory base address even if an I/O base address (see I/O Base Address, above) is also used. If a memory base address does apply, then the actual size of the memory segment used in the host PC's memory space may be from 2k to 64k bytes. If a Hypersignal-Macro Series software package is being used, see chapter 3 in the Hypersignal-Macro Series Hardware Reference Guide for information about memory base addresses for specific hardware types.

NOTE: If a memory base address is used, your system almost certainly will need the memory-mapped area to be "reserved" using the EMM386.EXE line in the system's config.sys file. This is necessary to avoid conflicts with memory-resident software programs and utilities, as well as other installed hardware. For more information about this, see Signalogic DSP Software System

Guide section 1.4.4, System Drivers and Resident Program Considerations, and EMM386.EXE Considerations (Reserving Memory Areas) in Signalogic DSP Software System Guide section 4.3.2, Memory-Mapped Transfer Gotchas.

Hardware Configuration Parameters

Board Designator

I/O Base Address

Memory Base Address

Processor Clock

Voltage Ranges

Number of Processors

DSP Program File

Analog Clock Type

Bus Type

Memory Architecture

Processor Clock

The Processor Clock field specifies the rate, in MHz, of the clock signal supplied to the DSP device on the selected DSP/analog hardware. This value is used by DSPower series and Hypersignal-Macro series software to calculate sampling frequency settings in cases where the DSP/analog hardware derives its sampling rate values from the processor clock.

NOTE: In some cases, the processor's speed may not be the same as its clock rate. For

example, DSP manufacturers may talk about an "100 MFLOPs device", which uses a 40 MHz clock input. Or they speak of a 40 MHz device, which actually takes a 20 MHz clock.

Hardware Configuration Parameters

Board Designator

I/O Base Address

Memory Base Address

Processor Clock

Voltage Ranges

Number of Processors

DSP Program File

Analog Clock Type

Bus Type

Memory Architecture

Voltage Ranges

The Voltage Ranges field sets the analog input and output voltage ranges for the selected DSP/analog hardware type. Entry is assumed to be in Volts and should be of the form:

```
inp,out
```

where inp and out are floating point values typically ranging from 0 to 10. For example, an entry of:

```
1.8,3.0
```

would specify a ±1.8V input voltage range, and a ±3V output voltage range. Note that this

parameter is significant only for hardware which has (software-selectable) programmable gain. In such cases, gain can be determined by comparing the entered value to that made in individual instrument and data acquisition functions. Otherwise, when hardware gain cannot be controlled via software, voltage range entry made in individual instrument and data acquisition functions have no effect on signal gain. In these cases, voltage ranges should exactly match the actual hardware settings, in order to produce correct amplitude units display. See chapter 3 in the Hypersignal-Macro Series Hardware Reference Guide for voltage range specifications for individual hardware types.

Hardware Configuration Parameters

Board Designator

I/O Base Address

Memory Base Address

Processor Clock

Voltage Ranges

Number of Processors

DSP Program File

Analog Clock Type

Bus Type

Memory Architecture

Number of Processors

The Number of Processors field reflects the number of processors

on the DSP/analog hardware type currently selected. This field cannot be changed in the Hardware Selector; if modification is necessary, the Hardware Setup File must be edited directly. See Number of Processors Field in Hardware Setup File, in section 8.3.1 below, Hardware Setup File Fields.

Hardware Configuration Parameters

Board Designator

I/O Base Address

Memory Base Address

Processor Clock

Voltage Ranges

Number of Processors

DSP Program File

Analog Clock Type

Bus Type

Memory Architecture

DSP Program File

The DSP Program File field specifies the default executable DSP file to download to the DSP/analog hardware if a file is not explicitly specified in a function, display, or instrument block in DSPower-Block Diagram software, or in DSLoadFileBoard and DSLoadFileProcessor calls in Hardware Library (included with DSPower-HWLib software).

The DSP Program File field should specify a file fully prepared for run-time initialization and execution, in the output format used by the DSP device manufacturer's linker utility. Typically, these linkers produce COFF (common object file format) files, which are handled automatically by the DSLoadxxx calls in Hardware Library.

The default filenames for this field are set to the files shipped standard with Hypersignal-Macro series software. For information about creating and modifying DSP executable files for use with Hypersignal-Macro Series software, see chapter 5, Using DSP Source Code Interface Software.

Hardware Configuration Parameters

Board Designator

I/O Base Address

Memory Base Address

Processor Clock

Voltage Ranges

Number of Processors

DSP Program File

Analog Clock Type

Bus Type

Memory Architecture

Analog Clock Type

The Analog Clock Type selection specifies the type of analog I/O clock (sampling frequency clock)

used on the DSP/analog hardware. This selection should be one of the following:

none

external

fixed, generated onboard or onchip

software programmable, generated onboard or onchip

NOTE: If the Analog Clock Type field is set to external or fixed internal, then you must remember to accurately enter sampling rate values in data acquisition, display, and instrument functions to exactly match the fixed value. Otherwise, axis and units readouts may be incorrect.

Hardware Configuration Parameters

Board Designator

I/O Base Address

Memory Base Address

Processor Clock

Voltage Ranges

Number of Processors

DSP Program File

Analog Clock Type

Bus Type

Memory Architecture

Bus Type

The Bus Type selection specifies the host computer bus type for the hardware. For most DSP/analog hardware, the "80286" or similar button should be selected, indicating that the bus interface on the hardware can handle 80286-type string move instructions at a minimum. Below are other Bus Type selections:

8086 instruction set only

80286 instruction set (string move instructions)

(future selections reserved for Signalogic)

Hardware Configuration Parameters

Board Designator

I/O Base Address

Memory Base Address

Processor Clock

Voltage Ranges

Number of Processors

DSP Program File

Analog Clock Type

Bus Type

Memory Architecture

Memory Architecture

The Memory Architecture field reflects the memory architecture of the DSP/analog hardware type

currently selected. This field cannot be changed in the Hardware Selector; if modification is necessary, the Hardware Setup File must be edited directly. See Memory Architecture Field in Hardware Setup File, in section 8.3.1 below, Hardware Setup File Fields.

The Memory Architecture field applies to onboard memory that is external to the DSP device on the DSP/analog hardware, and not necessarily to onchip memory. This field should indicate one of the following architecture types:

linear, both program and data in the same memory space

modified **Harvard** architecture, with program and data linear, but contained in separate memory spaces

modified Harvard architecture with program and data in separate memory spaces, and with data memory further subdivided into **vector** memory spaces

Hardware Configuration Parameters

Board Designator

I/O Base Address

Memory Base Address

Processor Clock

Voltage Ranges

Number of Processors

DSP Program File

Analog Clock Type

Bus Type

Memory Architecture

8.2.3 Hardware Values and Variables

[@j hardware value, hardware variable]

In DSPower series software, a hardware value can be initialized in two ways:

- when a hardware type is selected in the Hardware Selector and configured as needed
- when DSAssignBoard calls (and/or other calls) are made to the DSPower-HWLib DLL

There are as many hardware values possible as there are items in the hardware type listbox in the Hardware Setup file.

Hardware variables, on the other hand, are *logical references* to a classification of hardware, and may be one of several different values, and which may change in value dynamically. For example, a function might refer to the current "DSP Accelerator" hardware variable to attempt to execute all or

part of its task on installed DSP hardware. In this manner, the function does not care or know about the specific type of hardware installed, or whether it has changed.

Note that the current hardware value setting and hardware variable settings are not stored in the Hardware Setup File explained in section 8.3 below. They are instead stored in a separate private configuration file maintained by Hardware Manager (hwmgr.ini).

Hardware variables can be assigned using the "Set Hardware Variables" button in the Hardware Selector. Currently, this button is active only in DSPower-Block Diagram software. In DSPower-Block Diagram software, the Hardware Selector appears after selecting Configuration | DSP/Analog Hardware from the top-level worksheet menu. One use of hardware variables in DSPower-Block Diagram software is to allow operation of more than one type of DSP/analog hardware simultaneously when executing block diagrams.

Related Topics

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

8.3 Hardware Setup File

The Hardware Setup File is normally shipped in a default state listing all boards supported by Signalogic DSP Software components. It is an ASCII text file, with no special characters, and is installed by the Signalogic Software Setup Utility under the "xxx\hwlib" subdirectory as "hwsetup.lst", where "xxx" is the subdirectory specified during the DSPower series software installation (for example, "c:\dspower"). Currently, a single Hardware Setup File is maintained by Hardware Manager, regardless of the number of calling applications.

The Hardware Setup File is initially read into global memory from disk file upon the first instance of a call into Hardware Manager requiring data DSP/analog hardware configuration data. This occurs automatically, and includes the DSShowHardwareSelectorDlg call. Global memory is allocated as each new entry is read from the Hardware Setup File. If at some point, memory cannot be increased, NULL is returned and a subsequent call to DSGetHWMgrErrorStatus will return DSCOULDNOTINCREASEHWLISTMEM.

It is possible to change DSP/analog hardware configuration data by

calling the DSSetHWMgrEntry function with a pointer to a HWLISTENTRY structure. In this case, a DSWriteHWSetupFile call should be made if DSP/analog hardware configuration data has been changed, and needs to be saved. Note that if the Hardware Selector is invoked, and DSP/analog hardware configuration data is changed by the operator, followed by pushing the "OK" button, DSWriteHWSetupFile is executed automatically. For more information about accessing the Hardware Setup File and changing DSP/analog hardware configuration data, see section 8.3.2 below, Hardware Setup File Access, and HWLISTENTRY Structure in section 8.3.2.

Below is a partial segment of a typical Hardware Setup file, showing three (3) entries, where each entry is a line in the file. Note that the entries are shown with each line divided into two sections, due to the limited width of the printed version of the documentation.

Notes

1) Except for the board designator and driver ID fields, the second parameter in each field is the default and is not changed by Hardware Manager. If the user selects Restore from Defaults, this parameter is used to re-initialize the field. *The board designator and driver ID fields should not be changed under any circumstances. Doing so will cause undefined results.*

2) All parameters must be separated by at least two (2) spaces to be deemed separate.

3) Addresses are in hex ("0x" prefix or "h" suffix allowed).

4) The default DSP program files are set to executable DSP files shipped standard with Hypersignal-Macro series software, but can be changed as needed. For information about customizing Hypersignal DSP Source Code Interface files, see chapter 5, Using DSP Source Code Interface Software.

5) Board designator strings can be used directly in DSPower-HWLib DSAssignBoard and DSSetHWMgrCurVal calls. The DSShowHardwareSelectorDlg call returns a board designator string.

6) Driver IDs are subdivided into two parts: a major ID, and a minor ID. See section 8.2.3 below, Hardware Setup File: Driver IDs, for more information.

7) Entries in the Hardware Setup file are subdivided by manufacturer section names, such "[Ariel]", or "[Data Translation]", etc. Manufacturer section names occupy a line by themselves, and are used to cause alphabetic display in the hardware type listbox portion of the Hardware Selector dialog box (see section 8.2 above, Hardware Selector).

Whenever parameters are changed in the hardware type listbox or hardware configuration sections of the Hardware Selector (see section 8.2.2, DSP/Analog Hardware Configuration), and the OK button is clicked, changes are saved to the Hardware Configuration file. Changes are discarded if the Cancel button is clicked.

Related Topics

General Topics

Hardware Manager Access

Hardware Selector

Selecting DSP/Analog Hardware Types
DSP/Analog Hardware Configuration
Hardware Values and Variables
Hardware Setup File
Error Handling in Hardware Manager

8.3.1 Hardware Setup File Fields

This section contains a breakdown of individual fields in the Hardware Setup file. Note that these fields are a superset of the fields shown in the Hardware Selector dialog box, including all fields described in section 8.2.2 above, DSP/Analog Hardware Configuration. However, the Hardware Setup file includes several fields not displayed in the Hardware Selector dialog box. These are fields which cannot be changed dynamically, and are thus not user-selectable options. Examples include the description, driver IDs, number of processors, and memory architecture fields, which are fixed according to the design of the hardware.

Note that except for the description, board designator, and driver ID fields, the second parameter in each field is the "restore-from-default" parameter and cannot be changed from the Hardware Selector or from Hardware Manager calls. In the Hardware Selector, if the user selects the Restore from Defaults button, this parameter is used to re-

initialize the field. For an example and more detailed information, see the file excerpt and 1) under "Notes" in section 8.3 above, Hardware Setup File.

See below for explanations of the fields inside the Hardware Setup file.

Hardware Setup File Fields

Related Topics

Hardware Setup File Access

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

Error Handling in Hardware Manager

Description Field in Hardware Setup File

[@j description field]

This field describes the hardware type. Usually, the name or model of the hardware is given, sometimes also including a mention or abbreviation of the type of DSP device on the hardware. The description field should be in text

form, with no more than one (1) space separating words in the text, and should not exceed 50 characters.

Additional Hardware Setup File Fields

Board Designator

Driver IDs

Number of Processors

I/O Base Address

Memory Base Address

Bus Type

Analog Clock Type

Voltage Ranges

DSP Program File

Processor Clock

Memory Architecture

Board Designator Field in Hardware Setup File

[@j board designator field]

Entry of this field in the Hardware Setup file is essentially the same as its counterpart in the Hardware Selector. See Board Designator, above. Note that this field does not have a restore-from-default parameter. Invalid entry in this field, or entry which is not understood by a DSP Engine, will cause undefined results.

For a list of valid board designators, see either the Hardware Setup File (hwsetup.lst) or Signalogic DSP Software

Reference Guide section 3.2, Board Designator, Driver ID, and Classification Table.

Although board designators should be selected by selecting an item in the Hardware Selector listbox, they can also be entered directly. New hardware types, with new board designators, can also be added to the Hardware Setup file. This might be necessary in a case where a designator or valid suffix is not listed or correctly listed in the Hardware Setup file. Board designators are usually divided into three parts:

manufacturer abbreviation

DSP chip type

"subspecies" suffix

For example, a fictitious designator might be 'WID88', designating the Widget Systems DSP-88 board, containing the Arkansas Instruments DSP8800 device. A 'WID88-B' designator might then indicate the PC-88 board, which contains the less expensive (but also less powerful) DSP8802 device. In the latter case, the '-B' is a suffix indicating "subspecies" hardware, which usually involves a board similar in DSP device type, onboard memory type and amount, and digital interface, but different in analog I/O capability.

Additional Hardware Setup File Fields

Description

Driver IDs

Number of Processors

I/O Base Address

Memory Base Address

Bus Type

Analog Clock Type

Voltage Ranges

DSP Program File

Processor Clock

Memory Architecture

Driver ID Fields in Hardware Setup File

[@j driver ID field]

The driver ID fields specify hardcoded constants which indicate the type of hardware to DSP Engine drivers for DSP/analog hardware. The first field is the major driver ID, which indicates a "main hardware type" value. Major driver ID values are grouped according to DSP chip family type. The second field is the minor driver ID, which, if non-zero, indicates a hardware type closely related to the major type, but different in analog I/O or host PC interface characteristics.

NOTE: Grouping major ID values by DSP chip family type for various DSP/analog hardware must be maintained at all times. The major ID value is used by Hardware Library functions such as DSGetWordLength, DSDSPToIEEE, and

DSIEEEToDSP to determine the type of DSP chip being used.

For a list of driver ID values, see Signalogic DSP Software Reference Guide section 3.2, Board Designator, Driver ID, and Classification Table.

Additional Hardware Setup File Fields

Description

Board Designator

Number of Processors

I/O Base Address

Memory Base Address

Bus Type

Analog Clock Type

Voltage Ranges

DSP Program File

Processor Clock

Memory Architecture

Number of Processors Field in Hardware Setup File

[@j number of processors field]

This field indicates the number of processors on the hardware. Invalid entry in this field will cause undefined results.

Additional Hardware Setup File Fields

Description

Board Designator

Driver IDs

I/O Base Address

Memory Base Address

Bus Type

Analog Clock Type

Voltage Ranges

DSP Program File

Processor Clock

Memory Architecture

I/O Base Address Field in Hardware Setup File

[@j I/O base address field]

This field indicates the I/O base address, if applicable, for the hardware type. The restore-from-default half of this field should be left equal to the factory default setting for the hardware. See I/O Base Address, above, for more information.

Additional Hardware Setup File Fields

Description

Board Designator

Driver IDs

Number of Processors

Memory Base Address

Bus Type

Analog Clock Type

Voltage Ranges

DSP Program File

Processor Clock

Memory Architecture

Memory Base Address Field in Hardware Setup File

[@j memory base address field]

This field indicates the memory base address, if applicable, for the hardware type. The restore-from-default half of this field should be left equal to the factory default setting for the hardware. For more information, see Memory Base Address in section 8.2.2 above, DSP/Analog Hardware Configuration.

Additional Hardware Setup File Fields

Description

Board Designator

Driver IDs

Number of Processors

I/O Base Address

Bus Type

Analog Clock Type

Voltage Ranges

DSP Program File

Processor Clock

Memory Architecture

Bus Type Field in Hardware Setup File

[@j bus type field]

This field specifies the host computer bus type used by the hardware, and should be set to a '1' for most DSP/analog hardware. A '1' value indicates that the bus interface on the hardware can handle 80286-type string move instructions at a minimum. Below are valid Bus Type values:

8086 instruction set only

80286 instruction set (string move instructions)

Reserved for Signalogic

Additional Hardware Setup File Fields

Description

Board Designator

Driver IDs

Number of Processors

I/O Base Address

Memory Base Address

Analog Clock Type

Voltage Ranges

DSP Program File

Processor Clock

Memory Architecture

Analog Clock Type Field in Hardware Setup File

[@j analog clock type field]

The Analog Clock Type field specifies the type of analog I/O clock (sampling frequency clock) used on the DSP/analog hardware. This field should be one of the following:

none

external

fixed, generated onboard or onchip

software programmable, generated onboard or onchip

Additional Hardware Setup File Fields

Description

Board Designator

Driver IDs

Number of Processors

I/O Base Address

Memory Base Address

Bus Type

Voltage Ranges

DSP Program File

Processor Clock

Memory Architecture

Voltage Ranges Field in Hardware Setup File

[@j voltage ranges field]

The Voltage Ranges field sets the analog input and output voltage ranges for the selected DSP/analog hardware type. Entry is essentially the same as in the Hardware Selector. For more information, see Voltage Ranges in section 8.2.2 above, DSP/Analog Hardware Configuration.

Additional Hardware Setup File Fields

Description

Board Designator

Driver IDs

Number of Processors

I/O Base Address

Memory Base Address

Bus Type

Analog Clock Type

DSP Program File

Processor Clock

Memory Architecture

DSP Program File Field in Hardware Setup File

[@j DSP program file field]

The DSP Program File field sets the default executable DSP file for DSPower series functions to download to DSP/analog hardware if an explicit entry is not given. Entry is essentially the same as in the Hardware Selector. For more information, see DSP Program File in section 8.2.2 above, DSP/Analog Hardware Configuration.

Additional Hardware Setup File Fields

Description

Board Designator

Driver IDs

Number of Processors

I/O Base Address

Memory Base Address

Bus Type

Analog Clock Type

Voltage Ranges

Processor Clock

Memory Architecture

Processor Clock Field in Hardware Setup File

[@j processor clock field]

The Processor Clock field sets the value of the onboard clock rate being supplied to the DSP device on the DSP/analog hardware. Entry is in MHz, and is essentially the same as in the Hardware Selector. For more information, see Processor Clock in section 8.2.2 above, DSP/Analog Hardware Configuration.

Additional Hardware Setup File Fields

Description

Board Designator

Driver IDs

Number of Processors

I/O Base Address

Memory Base Address

Bus Type

Analog Clock Type

Voltage Ranges

DSP Program File

Memory Architecture

Memory Architecture Field in Hardware Setup File

[@j memory architecture field]

The Memory Architecture field specifies the type of memory architecture access used by the DSP device on the DSP/analog hardware. This field applies to onboard memory that is external to the DSP device, and not necessarily to onchip memory. This field should be one of the following:

linear, both program and data in the same memory space

modified Harvard architecture, with program and data in separate memory spaces

modified Harvard architecture with program and data in separate memory spaces, and with data memory further subdivided into vector memory spaces

Additional Hardware Setup File Fields

Description
Board Designator
Driver IDs
Number of Processors
I/O Base Address
Memory Base Address
Bus Type
Analog Clock Type
Voltage Ranges
DSP Program File
Processor Clock

8.3.2 Hardware Setup File Access

Hardware Manager (hwmgr.dll) contains several functions which can be used to access Hardware Setup File information from C/C++ and MATLAB programs. Below is a brief summary of some of these functions. For detailed information, see Signalogic DSP Software Reference Guide section 2.2.2, Hardware Manager Functions.

The two most basic functions are the DSReadHWSetupFile and DSWriteHWSetupFile calls. The former causes Hardware Manager to allocate global memory and read the Hardware Setup File into this memory, and the latter causes Hardware Manager to write the current memory contents out to the Hardware Setup File. If not yet

called, the DSReadHWSetupFile function is invoked be Hardware Manager functions which require DSP/analog hardware configuration data upon being called the first time, including the DSShowHardwareSelectorDlg call.

If a DSReadHWSetupFile call made from a calling application is successful, the return value is the global memory handle to a series of HWLISTENTRY entries containing DSP/analog hardware configuration data. The number of entries can be determined by calling DSGetHWMgrNumEntries. The global memory handle is created using the GMEM_SHARED attribute.

NOTE: It is not recommended that calling applications access HWLISTENTRY entries directly using the global memory handle returned by DSReadHWSetupFile. Instead, the DSGetHWMgrEntry call should be used.

The DSGetHWMgrEntry function can be used by a calling application to obtain a DSP/analog hardware configuration data. This function requires a pointer to a HWLISTENTRY structure, and an index indicating which entry to retrieve. Indexes can be found by using DSGetHWMgrEntryIndex, which returns the index matching a board designator string. Likewise, the DSSetHWMgrEntry function can be used by a calling application to store DSP/analog hardware

configuration data, presumably after modification or creation of a HWLISTENTRY structure. Note that in the case of DSSetHWMgrEntry, if the index parameter is equal to the total current number of entries, the entry is added rather than replacing an existing entry. The current number of entries can be obtained by using the DSGetHWMgrNumEntries function.

Other miscellaneous functions are also available. The DSGetHWMgrCurVal and DSSetHWMgrCurVal functions can be used to obtain and set the current Hardware Selector hardware type listbox selection. The DSGetHWMgrVar and DSSetHWMgrVar functions can be used to obtain and set Board Designators for a specified hardware variable (for more information, see section 8.2.3 above, Hardware Values and Variables). Finally, the DSShowHardwareSelectorDlg function will cause the Hardware Selector dialog box to be displayed (for more information, see section 8.2 above, Hardware Selector).

See Also

Related Topics

Hardware Setup File Access

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

Error Handling in Hardware Manager

HWLISTENTRY Structure

[@j HWLISTENTRY structure, HWLISTENTRY, PHWLISTENTRY]

Below is a reconstruction of the HWLISTENTRY structure, which is used in Hardware Manager calls such as DSGetHWMgrEntry and DSSetHWMgrEntry. The reprint below is divided into the following sections:

- information that is maintained in the Hardware Setup File
- information that is maintained internally by Hardware Manager, and is not stored in the Hardware Setup File

Each element in the structure is commented with a brief description. See also the notes below, after the structure definition, and section 8.3.1 above, Hardware Setup File Fields.

```
typedef struct {

                                                    //
                                                    comment


                                                    //
                                                    manufactu
```

rer
fie
ld

//
de
sc
ri
pt
io
n
fie
ld:
na
m
e/
m
od
el,
D
S
P
ty
pe
,
et
c.

//
bo
ar
d
de
si
gn
at
or
fie
ld

//
m
aj
or
D
S
P/
an
al
og
ha
rd
w
ar
e
dr
iv
er
ID
fie
ld

//
mi
no
r
D
S
P/
an

alog hardware driver ID field

// DSP program file field (current executable code file name)

// default DSP program file

// current I/O base address fie

ld
(i
n
he
x)

/ /
de
fa
ul
t
I /
O
ba
se
ad
dr
es
s

/ /
cu
rr
en
t
m
e
m
or
y
ba
se
ad
dr
es
s
fie
ld
(i
n
he
x)

/ /
de
fa
ul
t
ba
se
m
e
m
or
y
ad
dr
es
s

/ /
bu
s
ty
pe
fie
ld

/ /
de
fa
ul

```
t
bu
s
ty
pe


/ /
pr
oc
es
so
r
cl
oc
k
fie
ld
(s
pe
cif
ie
d
in
M
H
z)

/ /
de
fa
ul
t
pr
oc
es
so
r
cl
oc
k


/ /
vo
lta
ge
ra
ng
es
fie
ld
(s
ho
ul
d
be
en
te
re
d
as
'in
pu
t,
ou
tp
ut'
)

/ /
de
fa
ul
t
vo
```

ltage ranges

// analogclocktypefield: 0 = external clock, 1 = fixed internal, 2 = prog internal

// default analogclocktype

// memory architecture

field: "L" = linear, "H" = modified Harvard, "V" = vector

// default memory architecture

// number of processors field

// default number of processors

// following parameters not stored in Hardware Setup File

```
    // comment
    // calls-allowed classification
    // internal information used to mark pending updates, deletes, etc.
} HWLISTENTRY;

typedef HWLISTENTRY FAR* PHWLISTENTRY;
```

Notes

1) HWLISTENTRY structures should be completely initialized to zero before using, so default values can take effect for parameters not used. This can easily be accomplished using a call such as

```
_fmemset(&HWListEntry, 0, sizeof(HWLISTENTRY));
```

2) Supporting type definitions used above:

```
typedef char DESCSTR[50];
typedef char VALSTR[10];
```

Related Topics

Hardware Setup File Access

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

General Topics

Hardware Manager Access

Hardware Selector

Hardware Setup File

Error Handling in Hardware Manager

8.4 Error Handling in Hardware Manager

The Hardware Manager error handling scheme is straightforward. Most calls return a non-zero value on success, and a zero (NULL) on failure. When calls into hwmgr.dll fail, there are two levels of error possibilities: the DLL itself (or "base level"), and engine-related. The DSGetHWMgrErrorStatus and DSGetEngineErrorStatus calls can be used to attempt to learn more information about the most recent failure.

Note that the DSGetEngineErrorStatus call is actually inside enmgr.dll. Note also that Engine Manager maintains a "status listbox" which can be made visible using the DSShowEngMgrStatusWindow

call. This status window shows a history of *both status and error* text displays resulting from most Engine Manager and Hardware Manager calls. For complete information on error handling in Engine Manager, see section 7.5, Error Handling in Engine Manager.

Related Topics

General Topics

Hardware Manager Access

Hardware Selector

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

Hardware Setup File

Error Handling in Hardware Manager

8.4.1 DLL Error Handling in Hardware Manager

Hardware Manager has a "base level" error/status routine which can be used to look for general DLL error conditions not necessarily associated with engine problems. Below is an example showing further investigation after a DSSetHWMgrCurVal:

```
if (!DSSetHWMgrCurVal(szBoardDesignator)) {

    nErrorType = DSGetHWMgrErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSSetHWMgrCurVal would be NULL in the case of failure. Thus, to find out more information, the DSGetHWMgrErrorStatus call is used.

Below is a partial list of error values returned by DSGetHWMgrErrorStatus, along with a brief description of their meaning. These constants can be found mostly in the hwmgr.h file and partly in the dllerr.h file; for a complete list all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross-Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions.

dllerr.h or hwmgr.h constant	Summary of Meaning
DSCOULDNOTINITIALIZEDLL	

DSUNABLETOREADHWSETUPFILE

DSCOULDNOTLOCKHWLIST

DSBOARDDESIGNATORNOTLISTED

DSCOULDNOTFINDHWSETUPFILE

DSCOULDNOTINCREASEHWLISTMEM

Note that in C/C++ programs, it is safest to treat the above error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions, for more information about this grouping.

General Topics

Hardware Manager Access

Hardware Selector

Selecting DSP/Analog Hardware Types

DSP/Analog Hardware Configuration

Hardware Values and Variables

Hardware Setup File

Error Handling in Hardware Manager